NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

DISSERTATION

AN INFORMATION-CENTRIC APPROACH TO AUTONOMOUS TRAJECTORY PLANNING UTILIZING OPTIMAL CONTROL TECHNIQUES

by

Michael A. Hurni

September 2009

Dissertation Supervisor: I. Michael Ross

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2009	**3. REPORT TYPE AND DATES COVERED** Dissertation	
4. TITLE AND SUBTITLE: An Information-centric Approach to Autonomous Trajectory Planning Utilizing Optimal Control Techniques			**5. FUNDING NUMBERS**
6. AUTHOR(S) Michael A. Hurni			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**
13. ABSTRACT (maximum 200 words)			

This work introduces a new information-centric pseudospectral optimal control-based algorithm for autonomous trajectory planning and control of unmanned ground vehicles with real-time information updates. It begins with a comprehensive study and comparison of the various path planning methods currently in use. It then provides an analysis of the optimal control method, including vehicle and obstacle modeling techniques, several different problem formulations, and a number of important insights on unmanned ground vehicle motion planning. The new algorithm is then utilized on a collection of motion planning scenarios with varying levels of information; the performance of the planner and the solution accuracies under these varying levels of information are studied for both single and multi-vehicle scenarios. The multi-vehicle scenarios compare and contrast centralized, decentralized, decoupled, coordinated, cooperative, and prioritized control methods. Finally, the versatility of the planner (and the optimal control technique) is demonstrated, as it is used as both a path follower and trajectory planner in a collection of scenarios, including multi-vehicle formations and sector keeping.

14. SUBJECT TERMS Optimal Control, Pseudospectral, Autonomous Trajectory Planning, Unmanned Ground Vehicles, Real-Time, Path Planning, DIDO			**15. NUMBER OF PAGES** 297
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

AN INFORMATION-CENTRIC APPROACH TO AUTONOMOUS TRAJECTORY PLANNING UTILIZING OPTIMAL CONTROL TECHNIQUES

Michael A. Hurni
Commander, United States Navy
B.S. in Electrical Engineering, University of New Hampshire, 1989
M.S. in Mechanical Engineering, Naval Postgraduate School, 1997

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN MECHANICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2009

Author: Michael A. Hurni

Approved by:

I. Michael Ross
Professor of Mechanical and Astronautical Engineering
Dissertation Supervisor & Committee Chair

Fotis A. Papoulias
Professor of Mechanical and Astronautical Engineering

Isaac I. Kaminer
Professor of Mechanical and Astronautical Engineering

Wei Kang
Professor of Mathematics

Xiaoping Yun
Professor of Electrical and Computer Engineering

Approved by: Knox Millsaps, Chair, MAE Department

Approved by: Douglas Moses, Vice Provost for Academic Affairs

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This work introduces a new information-centric pseudospectral optimal control-based algorithm for autonomous trajectory planning and control of unmanned ground vehicles with real-time information updates. It begins with a comprehensive study and comparison of the various path planning methods currently in use. It then provides an analysis of the optimal control method, including vehicle and obstacle modeling techniques, several different problem formulations, and a number of important insights on unmanned ground vehicle motion planning. The new algorithm is then utilized on a collection of motion planning scenarios with varying levels of information; the performance of the planner and the solution accuracies under these varying levels of information are studied for both single and multi-vehicle scenarios. The multi-vehicle scenarios compare and contrast centralized, decentralized, decoupled, coordinated, cooperative, and prioritized control methods. Finally, the versatility of the planner (and the optimal control technique) is demonstrated, as it is used as both a path follower and trajectory planner in a collection of scenarios, including multi-vehicle formations and sector keeping.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank my dissertation committee for all their time and efforts in helping me achieve this important milestone in my life. But I especially want to thank my dissertation supervisor, Dr. Ross, for giving me the latitude to work independently on my research. His advice and direction at our weekly meetings were invaluable to the advancement of my research and writing skills.

Without the guidance of Pooya Sekhavat, the completion of this dissertation would have been considerably more difficult. His knowledge and experience in the field helped immeasurably. When I got stuck, I sought him out.

Lastly, Qi Gong and Mark Karpenko bear considerable credit for advising me throughout this educational process.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Autonomous trajectory planning of unmanned vehicles has been one of the main goals in robotics for many years. Recently, this problem has become particularly important as a result of rapid growth in its applications to both military and civilian missions. There has been much research and development in trajectory planning over the years, but only recently have advancements in mathematics, improved algorithms and faster processor speeds made optimal control techniques viable for the conduct of real time trajectory planning. Research scientists have known for many years that optimal control techniques could be used for trajectory planning, but only in the last few years has it become possible to consider such an avenue for real time applications. For these reasons, trajectory planning using optimal control techniques is still in its infancy.

In [1], it is shown how optimal control methods can accomplish the path planning and motion control problems in unmanned or robotic systems and simultaneously improve system autonomy. It also shows that these techniques hold the potential for mathematically optimal solutions while possessing the ability to account for dynamic and kinematic constraints, a characteristic that other techniques cannot demonstrate. However, it only scratched the surface of trajectory planning using optimal control, and cited the fact that much research still needs to be conducted before any real-world testing of trajectory planning can be conducted. It also cited the need for analyzing these techniques for use with cooperative motion between multiple vehicles.

The focus of this work is in the development of an information-centric algorithm for the conduct of autonomous trajectory planning of an unmanned ground vehicle using optimal control techniques. The algorithm is used to study the effects of varying levels of information on the trajectory planning problem and resulting solution for both single and multi-vehicle scenarios. This algorithm provides the dynamics by which a static (open loop) optimal control problem formulation can be used to solve a series of optimal control problems in real time in a closed loop form. In other words, the overall dynamic optimal control problem (closed loop/real time) is simply a collection of open loop (static) optimal control problems performed over and over in time. The dynamic problem can be considered as an outer loop, which decides from one iteration to the next, what the static problem formulation will be.

This dissertation is laid out as follows. Chapter II is a comprehensive study and comparison of the various path planning methods in use today. Chapter III is an analysis of the optimal control trajectory planning method, and includes several different problem formulations and issues on unmanned ground vehicle motion planning. It includes the vehicle and obstacle modeling techniques and all the verification and validation steps that must be performed to prove the solutions are feasible. Chapter IV presents the development and implementation of a new information-centric pseudospectral (PS) optimal control-based algorithm for autonomous trajectory planning and control of unmanned ground vehicles with real-time information updates. Chapter V utilizes the algorithm on a collection of motion planning scenarios with varying levels of information and studies the performance of the planner and the solution accuracies under these varying levels of information. Chapter VI shows the portability of the algorithm by demonstrating its use as a path follower. It demonstrates the flexibility of the technique through its ability to be utilized simultaneously for path following and trajectory planning. Chapter VII extends the work to multiple vehicles. It solves various multi-vehicle scenarios, including control methods that are centralized, decentralized, decoupled, coordinated, cooperative and/or prioritized. The quality of the solutions and the performance of the algorithm are again studied under varying levels of information, but this time the level of cooperation and prioritization between vehicles is also varied. Finally, Chapter VIII provides the conclusions and future work.

II. TRAJECTORY PLANNING METHODS

A. BACKGROUND

The purpose of the analysis in this chapter is to perform a preliminary study of the various trajectory planning methods so as to confirm the choice of which method is best to pursue in further study. It is important here to distinguish between *path planning*, *motion planning* and *trajectory planning*. *Path planning* finds a feasible path from start to goal. When that computed path is parameterized by time, it becomes *motion planning*. When the control solution (actual commands to the system actuators) for the computed path is determined, along with the time parameterization, it becomes *trajectory planning* [2]. There are two approaches to trajectory planning for a dynamic system: The decoupled approach and the direct approach. The decoupled approach involves first searching for a path (using a path planner) and then finding a time-optimal time scaling for the path subject to the actuator limits. The direct approach searches for the trajectory directly within the system's state space [3]. The best trajectory planning method is the one that is the most versatile, requiring the least amount of modifications to perform the desired tasks. It is shown in this chapter that the optimal control method of trajectory planning will be the best suited at meeting all the desired performance parameters. The focus of this research will be in studying trajectory planning of Unmanned Ground Vehicles (UGVs) utilizing optimal control techniques.

B. PROBLEM FORMULATION AND ANALYSIS

This work studies the real-time autonomous trajectory planning of UGVs utilizing optimal control techniques. UGVs will be referred to simply as *vehicles* from here on out. The terms "real-time" and "optimal" are merely desired criteria in this work for any method that is chosen to accomplish the task of trajectory planning. In the decoupled trajectory planning methods, path planning is the basic problem at hand. *Path planning* is defined, in [2], as identifying a trajectory that will cause a robot to reach its goal location when executed. Notice that in the above definition, the words "path planning" and "trajectory" appear in the same sentence. The words "path planning" and "trajectory planning" become blurred in the literature because of the fact that most of the more popular methods use path planners to conduct trajectory planning by either using the decoupled approach or by adapting what is traditionally a path planner to search in the vehicle's state space to perform trajectory planning. For these reasons, path planning and trajectory planning are often used synonymously in the literature and in this work as well. So how will this trajectory planning problem be accomplished? What criteria will have to be followed? What parameters will have to be met?

1. Benchmark Problem

No one benchmark problem can be used universally to describe the utility of all the path/trajectory planning methods to be discussed. Figure 1 shows the initial problem to be used in discussions concerning each planner. It is a simple obstacle avoidance problem. One vehicle has a start point, a goal, and two obstacles (labeled #1 and #2) located within a set environment. The boundary of the environment can either be an imaginary boundary that the vehicle cannot cross, or it could be an actual boundary such as the walls of a room. Regardless, the vehicle cannot leave the boundaries set by the problem as it travels to its goal.

Once the initial problem is discussed, various problem scenarios will be generated (depending on the path planning method) to help facilitate understanding the utility of the method being discussed.

2. Parameters and Criteria Considered

Each path/trajectory planning method has its advantages and disadvantages. As each method is discussed, its utility will be graded over the required parameters/criteria listed below. An 'A' will indicate the method can achieve the desired performance in that area. A 'C' indicates the method is unable to achieve the desired performance without help, which can come in various forms. For example, help could be in the form of a trajectory following subsystem to track the planned trajectory. An 'F' indicates the inability of a method to perform as desired. In path planning, it may be able to reach the goal in certain situations, but if conditions exist that prevent the method from achieving success, then the method will not work, and a grade of 'F' is awarded. It must be noted here that different versions of a planner may need to be used to achieve success in different performance parameters. This means that even if a planner gets high grades across the board, it may still not be a good planner if we cannot achieve success in the various parameters simultaneously.

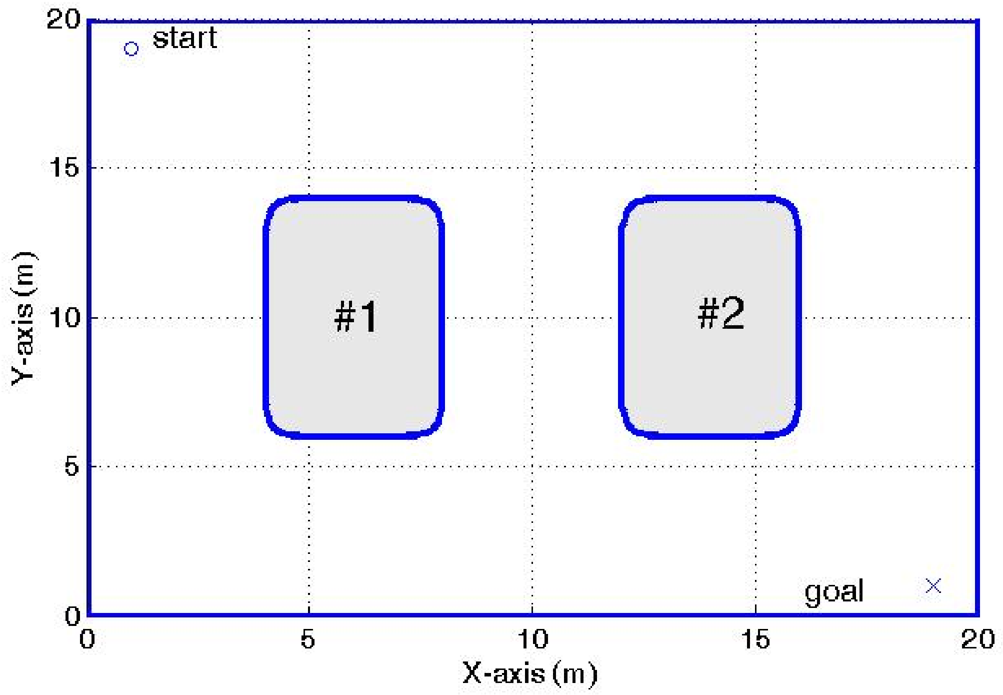


Figure 1. The Benchmark Problem.

- Path Planning and Control. Can the method being used not only solve for a path to the goal but also determine the control necessary to reach the goal? This is important, because if a path is achieved with no control solution, then the vehicle would require the addition of a trajectory following subsystem.

- Optimality. Is the path to the goal optimal and in what sense is it optimal? Can the planning method achieve optimality for not just minimum time problems, but also be able to optimize other criteria, such as fuel cost or some vehicle parameter? It will not always be desired to get from some start point to the goal in the minimum time possible. Furthermore, going the minimum distance does not always achieve minimum time. Since optimality is not the only criteria being measured, a "near optimal" solution will be sufficient to meet the requirements of this parameter.

- Obstacle Avoidance. Can the method account for more than just static obstacles that were originally accounted for at the start of the problem? *Obstacle avoidance* is defined, in [2], as modulating the trajectory of a robot in order to avoid collisions. The key here is reacting to unforeseen events (obstacles not known *a priori*) while maintaining some degree of optimality in the vehicle trajectory. Can it still solve the problem while accounting for the dynamics of moving obstacles? This becomes extremely important in a multiple vehicle environment, as it is common to treat other vehicles as moving obstacles.

- Handling Vehicle Constraints. Can the method account for vehicle constraints, such as turning radius or max/min velocity? The velocity constraint preventing instantaneous sideways motion is called a *nonholonomic* constraint, and the path planner must take this into account [3].

- Global vs. Local Information. Does the method operate on global information, local information, or both? In other words, is the planner information centric? Global information is necessary in path planning; without it, the vehicle would simply head straight at the goal and conduct obstacle avoidance on the way. Without local information, a collision would occur if the vehicle encountered an obstacle not originally accounted for in the initial trajectory. The vehicle must be able to see a new (not yet considered in the solution) obstacle and find a new path around it to the goal. Ideally, the best methods would plan the initial trajectory with global information, then adjust the path as local information is obtained. This path adjustment should be conducted by completely recalculating the trajectory from the current vehicle location to the goal, thus preserving solution optimality. This can only work if the path planner is information centric. In other words, it doesn't just get the vehicle around the next obstacle in its way; it is always taking into account the *big picture*.

- Computational Complexity. Can the vehicle solve the path problem quickly enough to be able to use the planner in real-time while the vehicle is in motion to the goal? This is important, because the environment will change (new local information obtained) as the vehicle moves toward its goal. It can also go hand in hand with the vehicle's ability to account for errors and uncertainties, as discussed below.

- Portability. Can the method be adapted to new obstacle environments or even totally different dynamical systems? For example, could the same method being used to plan a path for a reconnaissance vehicle through a neighborhood of buildings be used to steer a mine hunting vehicle through a mine field in a zigzag pattern? Can the method be adapted from a two dimensional problem to a three dimensional problem? Could the same method used on an UGV be used to land an airplane safely?

- Completeness [2]. Can the method guarantee a successful solution to all possible problems, if one exists? If a solution does not exist, can it report as such?

- Multiple Vehicles. We assume that if the planner can handle moving obstacles, then planning paths for multiple vehicles in a decentralized (decoupled) manner is automatically possible by treating other vehicles as moving obstacles. The issue here is in determining the ability of each method to prioritize vehicles, to plan with varying levels of cooperation among vehicles, and finally, to conduct centralized planning (i.e., multiple vehicles treated as one system with many state variables).

- Handling Errors and Uncertainties. These errors and uncertainties come from approximations in the modeling of the vehicle, errors in sensor data and accuracy, and external unforeseen forces. Either the vehicle must use feedback along with some trajectory following subsystem to reach its goal based on the original *offline* solution, or the vehicle continuously updates the path in real-time to account for its positional error off the previous trajectory to prevent these errors from blowing up as the vehicle moves along in time. This latter solution goes hand in hand with Computational Complexity.

C. DISCUSSION OF PATH/TRAJECTORY PLANNING METHODS

The various path/trajectory planning methods are discussed in detail in [2] and [3]. The following sections discuss each method and grade them according to the desired criteria.

1. Bug Algorithms

Bug Algorithms are known more as obstacle avoidance algorithms than path planners. They were chosen here to represent the example of obstacle avoidance systems due to their simplicity and the fact that they are being used successfully in mobile robotics today. Obstacle avoidance algorithms are being covered here for completeness and because they are often used in concert with other planners to form better hybrid approaches to motion planning. There are numerous other obstacle avoidance algorithms and techniques. They include Vector Field Histograms, the Bubble Band Technique, Curvature Velocity Techniques, the Schlegel Approach, the ASL Approach and many more. The details of these other methods are outlined in [2].

Bug algorithms are described, in [3], as being among the earliest and simplest sensor-based planners. In fact, being sensor-based and using only the immediate data, Bug algorithms are referred to as obstacle avoidance algorithms in [2]. We can, however, still call it a planner because it does start out with a rudimentary plan: It knows its start point and its goal, and its initial plan is to move in a straight line to the goal. The vehicle is assumed to be a point operating in the plane with a contact sensor or a range sensor to detect obstacles. The algorithms are straightforward to implement, requiring 2 behaviors: Moving in a straight line toward the goal and boundary following. The direction the vehicle goes when encountering an obstacle is entirely arbitrary. The vehicle can be told to randomly follow the obstacle on its right or on its left, or it can be told to always follow obstacles on the same side. The bottom line is the vehicle does not have enough information about the obstacle to know which direction would be optimal.

The Bug1 algorithm assumes the vehicle has a contact sensor or zero range sensor that can detect an obstacle boundary if the *point* vehicle *touches* it. The vehicle moves in a straight line toward the goal until it either encounters the goal or it runs into an obstacle. If the vehicle encounters an obstacle it circumnavigates it while calculating the optimal leave point from the obstacle, which is the point on the obstacle closest to the goal. Once the circumnavigation is complete, the vehicle follows the boundary to the leave point and then resumes a straight line path toward the goal. Figure 2 shows the trajectory of a vehicle using the Bug1 algorithm in the benchmark problem. Figure 3 shows an example of the vehicle being trapped inside an obstacle. Once the vehicle tries to move towards the goal at the leave point, it re-encounters the obstacle. This behavior would result in the algorithm returning a message that the goal cannot be achieved.

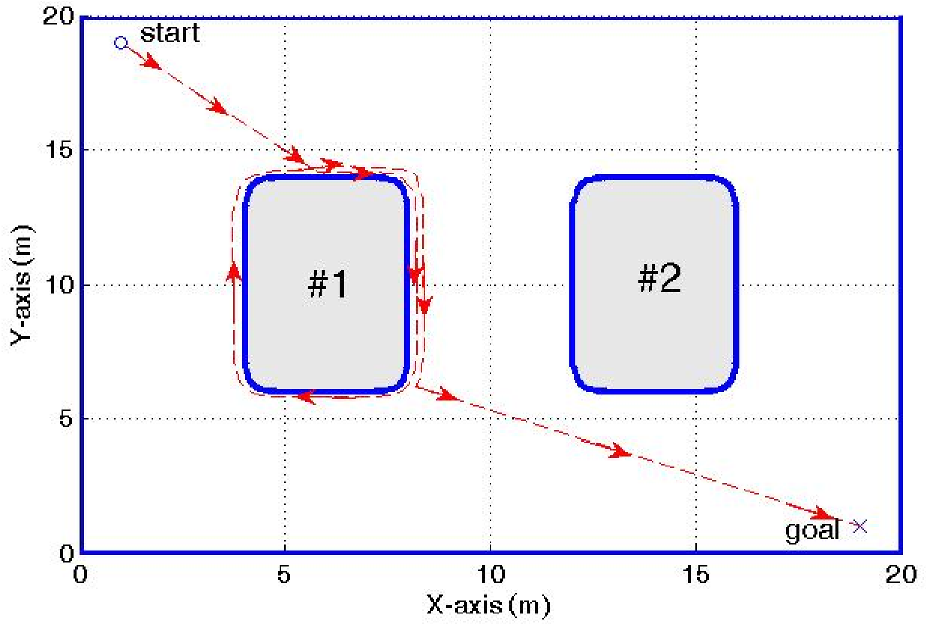


Figure 2. Bug1 Algorithm in the Benchmark Case.

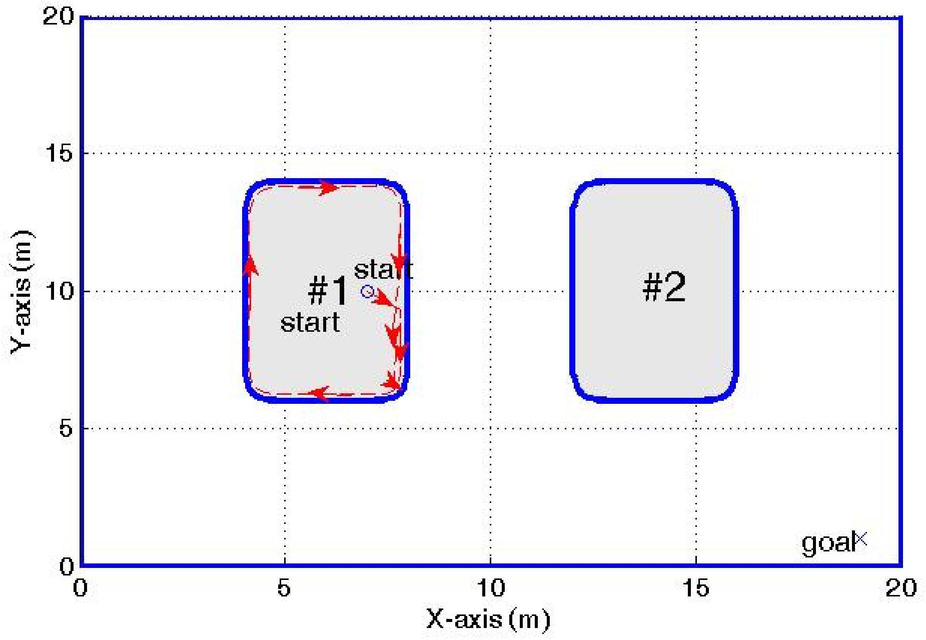


Figure 3. Bug1 Algorithm with Vehicle Trapped Inside Obstacle.

The Bug2 algorithm differs from Bug1 in that it remembers the original straight line path from start to goal. When the vehicle encounters an obstacle, it follows the obstacle's boundary until it re-encounters the original straight line path. If the vehicle is closer to the goal than when it started boundary following, it will return to the original path. Figure 4 demonstrates this behavior for a vehicle that knows which way to turn. In reality the direction of the turn would be arbitrary or always in the same direction. In a vehicle that always turns left, it would follow obstacle #2 over and around the top, which is far less than optimal. If the vehicle re-encounters its original departure point, the algorithm returns a message that there is no solution.

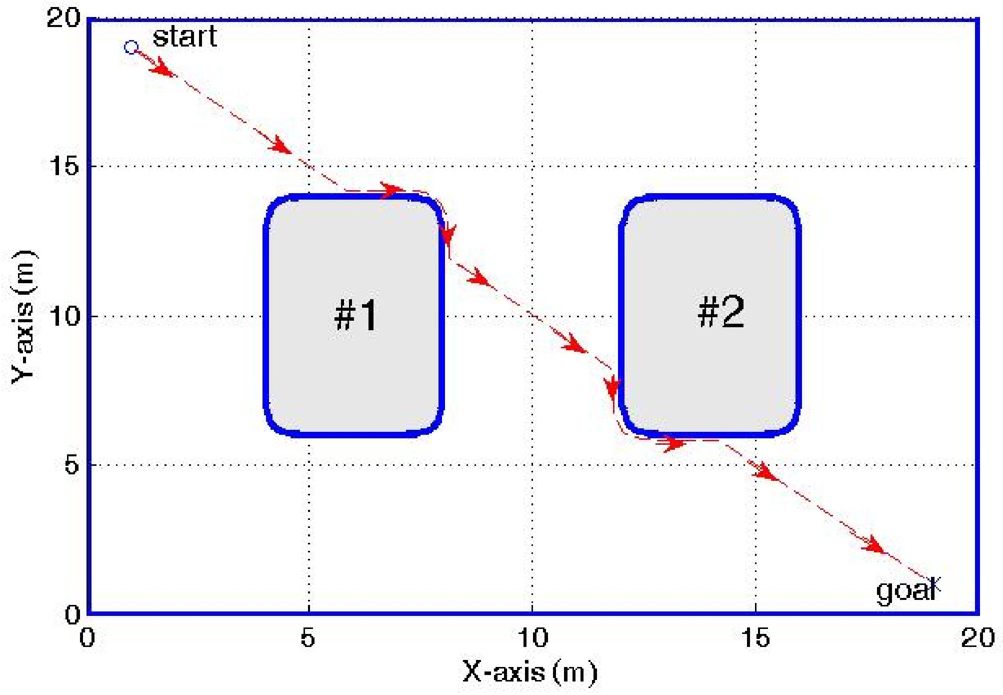


Figure 4. Bug2 Algorithm in the Benchmark Case.

The TangentBug algorithm uses finite range sensors to find shorter (more near optimal) paths to the goal. The vehicle again travels on a straight line path to the goal until it detects an obstacle. As long as the sensed portion of the obstacle does not impede the vehicle's path, it will continue to move straight at the goal. Once the sensed portion of the obstacle impedes the vehicle's path, it will alter its path to follow a tangent to the obstacle that minimizes the distance the vehicle needs to travel. In this case, the vehicle (or the algorithm) determines which way to turn based on which tangent direction will result in the shortest distance to goal. The vehicle continues to alter its course to stay tangent to the obstacle until the distance to the goal can no longer be decreased, at which point the vehicle switches from *motion toward goal* to *boundary following*. It will follow the boundary until the *reach distance* is less than the *followed distance* [3], at which point it switches back to *motion toward goal* behavior. The *reach* is the distance between the goal and the closest point on the followed obstacle that is within line of sight of the vehicle. The *followed* is the shortest distance between the boundary which had been sensed and the goal. Figures 5, 6 and 7 show the trajectories of a vehicle using the TangentBug algorithm with zero range, finite (*1 meter*) range and infinite range sensors respectively.

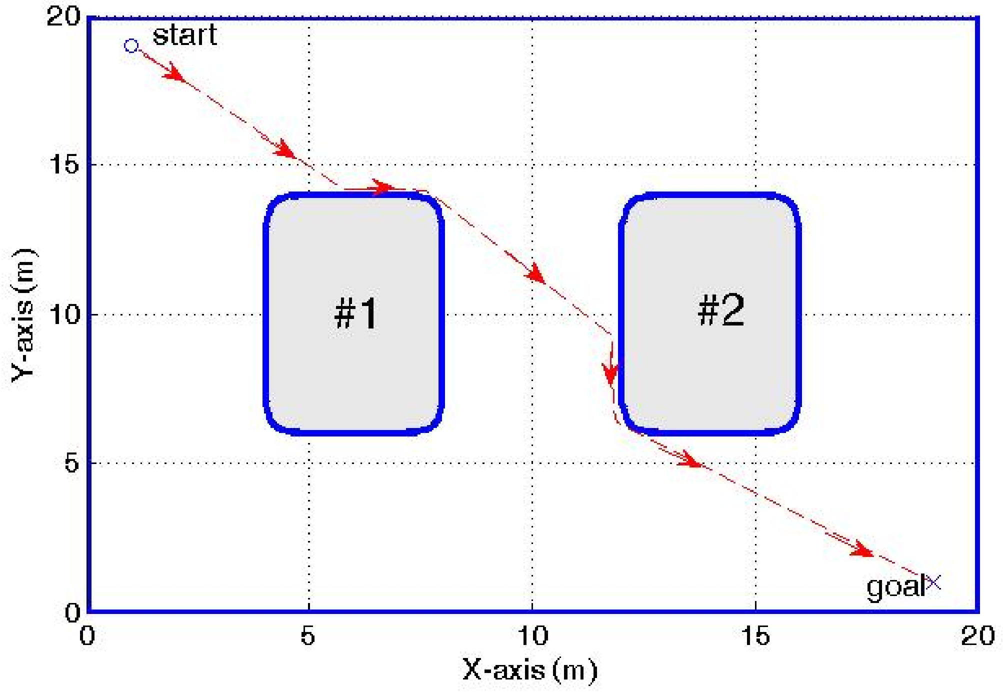


Figure 5. TangentBug Algorithm in the Benchmark Case (zero range sensor).

There are numerous other variations of Bug algorithms that have been experimented with in order to improve on the Bug1, Bug2 and TangentBug versions. Some show improvement in limited areas, but have disadvantages as well.

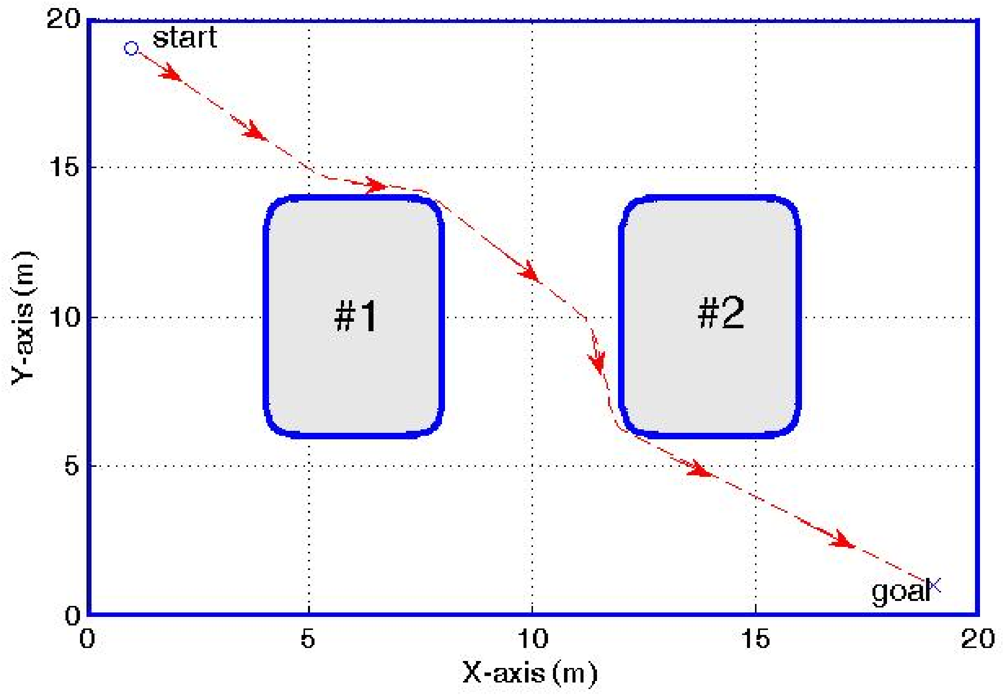


Figure 6. TangentBug Algorithm in the Benchmark Case (*1m* range sensor).

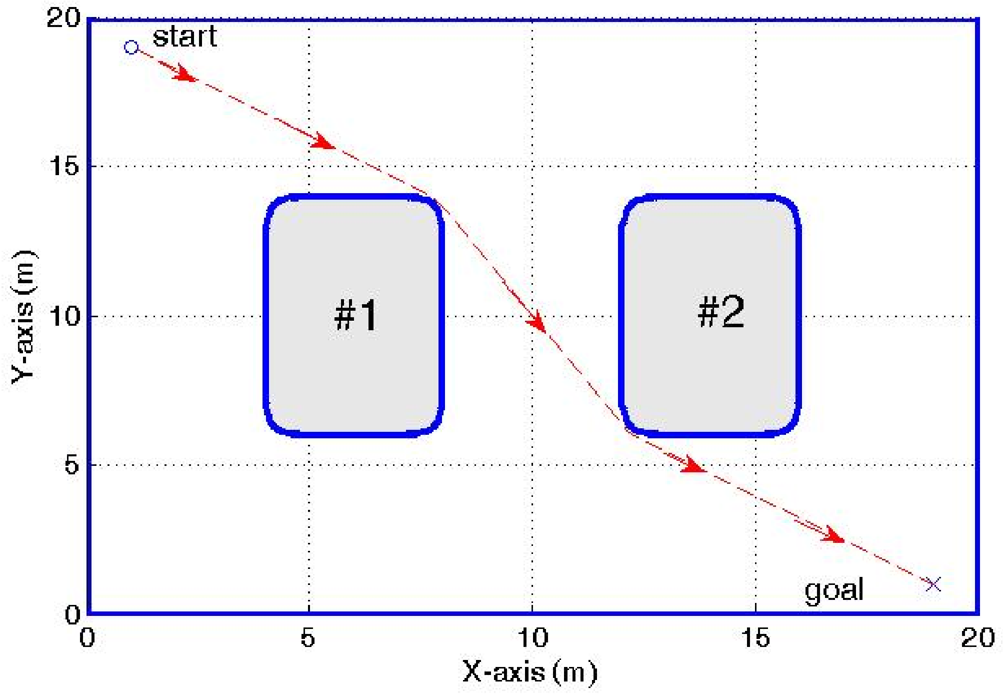


Figure 7. TangentBug Algorithm in the Benchmark Case (infinite range sensor).

In [4], a Bug algorithm is introduced that is a combination of the TangentBug and Bug2 in order to extend these purely static algorithms to a dynamic environment.

Range-sensor based navigation in three dimensions is introduced, in [5], by combining a two dimensional Bug algorithm with a three dimensional surface exploration algorithm. The 3DBug algorithm navigates a point vehicle in a three dimensional unknown environment using position and range sensors. It uses two basic behaviors: Moving toward the goal and moving along an obstacle surface.

VisBug and DistBug [6] are proposed improvements (using range sensors) in the Bug2 algorithm which enhance the condition the vehicle uses to stop boundary following and resume movement toward the goal. These improvements generate shorter paths to the goal.

K-Bug [6] is an algorithm that solves the problem of which direction the vehicle should go when encountering an obstacle. This method determines the furthest visible point of the obstacle in each direction and drives the vehicle toward the closer of these visible points; this is a waypoint. It also does not require the vehicle to re-connect with the original path to goal as in Bug2, but rather generates a new path to goal at each waypoint. Once it reaches the waypoint, it again determines the furthest visible points on the obstacle. Once the vehicle has a clear path, it goes back to motion toward goal. Of course, like the other Bugs, scenarios exist that cause K-Bug to guide the vehicle along a path far less than desirable. Furthermore, if the obstacle size is larger than the vehicle's sensor range, the vehicle may not pick the best direction.

CautiousBug [7] executes a conservative spiral search in both directions of the obstacle boundary to determine which direction the vehicle should take. This allows the vehicle to explore both directions before determining the best path. Of course, in some cases this results in a lot of additional path length being covered on the vehicle's way to the goal.

- Path Planning and Control. The vehicle will require some form of feedback control in order to maintain its straight line path to the goal. It will also require additional algorithms and control to allow it to conduct boundary following. With that additional software, Bug algorithms can accomplish this task. Grade: C.

- Optimality. Bug algorithms do not provide optimization. Figure 7 shows that in some cases, the TangentBug can provide a shortest distance solution to the goal, however, that was only due to a favorable geometry and the assumption that the sensor had infinite range. When the vehicle determines a tangent to an obstacle, it does this by computing discontinuities in the range sensors' outputs. This results in the vehicle traversing a path tangent to every obstacle between it and the goal, which is not the most efficient solution in many cases. Figure 8 demonstrates how different obstacle positioning and size results in a less than optimal trajectory, even while still assuming the sensor has infinite range. Grade: F.

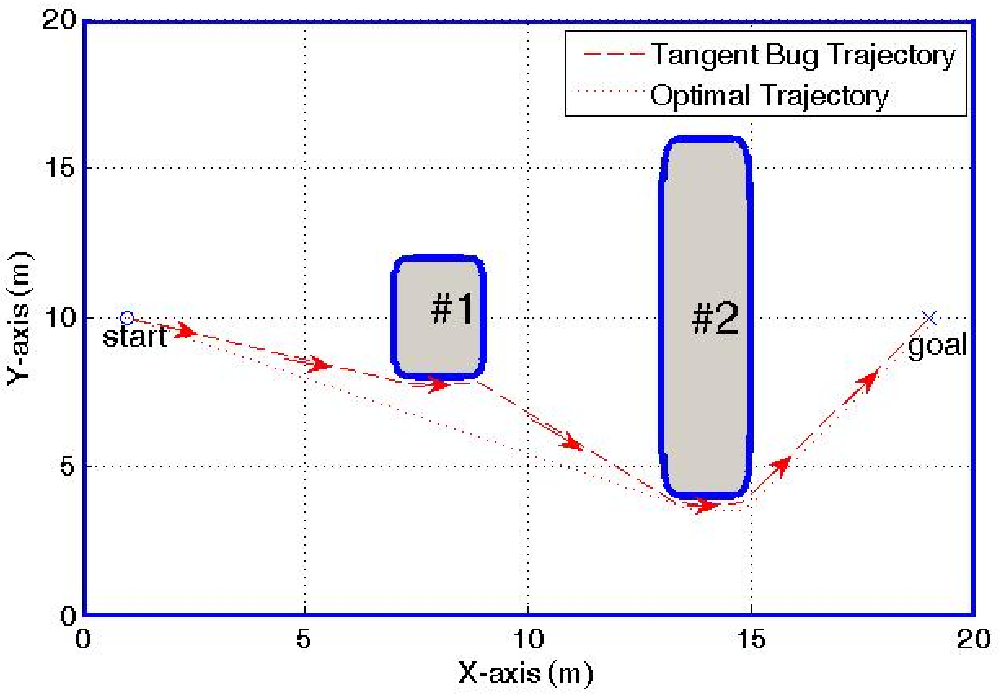


Figure 8. TangentBug's Less than Optimal Trajectory.

- Obstacle Avoidance. Bug Algorithms are specifically designed for obstacle avoidance; however, their viability breaks down when used in dynamical environments. The TangentBug/Bug2 variant used in [4] shows the method works for simple dynamic situations, however, more complicated dynamical environments were not studied, and no collision risk analysis was conducted. The work concludes by suggesting prediction be added to the Bug algorithm, taking into account the velocities of obstacles. Without some kind of prediction it is obvious that if obstacle speed is greater than vehicle speed, collisions would be inevitable. Finally, the boundary following behavior of Bug algorithms is not viable if the obstacle is moving. Grade: F.

- Handling Vehicle Constraints. Vehicle kinematics/dynamics and *nonholonomic* constraints are not accounted for in this method. Additional techniques are needed to account for vehicle velocity and constraints such as turning radius. It is necessary to smooth out the planned path to form a trajectory that an under actuated vehicle can follow while staying inside its physical limitations. This task can be accomplished with the addition of the necessary control software and algorithms as mentioned above in Path Planning and Control. Grade: C.

- Global vs. Local Information. The only global information that Bug algorithms use is the fact that they know the start point and the goal. Other than that, they are purely reactive, relying on immediate local information obtained by the vehicle's contact or range sensors. An improvement to this method would be to use it in concert with another planning method that takes global information into account. The other method would be used initially *offline* to plan the path for the vehicle's trip; then the Bug algorithm could be used in real-time to avoid obstacles on the way. Of course, the Bug algorithm would have to be adjusted from *motion toward the goal* to *motion toward the next waypoint*, as determined by the planned trajectory. Handling moving obstacles would still be a problem, and solution optimality would not be preserved. Grade: C.

- Computational Complexity. As discussed earlier, this is one of the simplest sensor-based planners, designed to be used real-time. Grade: A.

- Portability. Bug algorithms are attractive in their simplicity; however they have a limited range of use. It would not be able to land an airplane safely or guide a vehicle in any kind of pattern other than to avoid obstacles. It is shown, in [5], that work has been conducted in three dimensions and a viable algorithm (3DBug) has been generated. The method can (but not without much help) be adapted for use in any environment for its originally designed purpose of obstacle avoidance. Grade: C.

- Completeness. Not all Bug algorithm versions can guarantee a successful solution to all possible problems. However, taken as a whole, there is always a version that will be able to solve a given problem. For example, Figure 9 shows that Bug2 fails miserably in the *open door* example if the vehicle turns the wrong way. TangentBug and K-Bug will similarly pass or fail depending on the position of the open door relative to the obstacle location. However, CautiousBug would successfully achieve the goal at the cost of traveling a far longer than optimal distance. So as a whole, Bug algorithms will guarantee a solution, if one exists, and report if a solution does not exist. Grade: A.

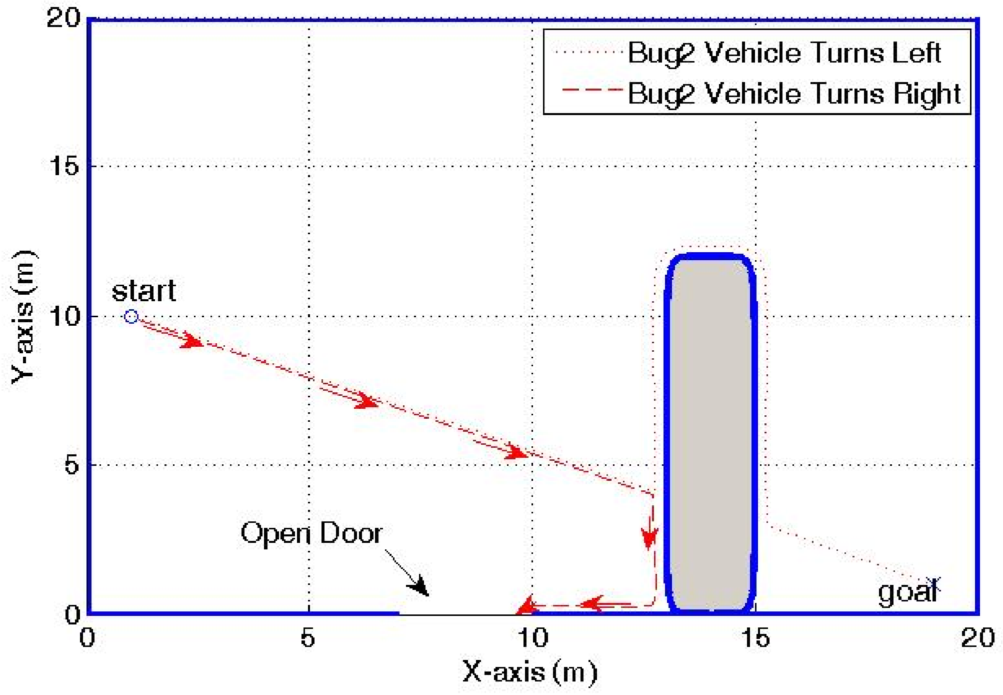


Figure 9. Bug2 Algorithm in Open Door Concept.

- Multiple Vehicles. This method can not conduct path planning for multiple vehicles due to its inability to plan paths when confronted with moving obstacles. (see Obstacle Avoidance). Grade: F.

- Handling Errors and Uncertainties. Bug algorithms are described, in [3], as assuming the vehicle is a point with perfect positioning (no positioning error). Assuming the vehicle has the means to periodically update its position (such as GPS), it will need some kind of feedback control in order to update its path to the goal. Grade: C.

Suffice it to say that obstacle avoidance algorithms, by themselves, do not make good path planners, at least not given the performance requirements of this work. However, they provide a strong tool for obstacle avoidance when used in concert with other path planning methods. The best solution when using a real-time reactive planner such as those listed above is to integrate it with an offline global path planner.

2. Artificial Potential Fields

This approach was originally invented for robot manipulator path planning and is used often and under many variants in the mobile robotics community [2]. Artificial Potential Fields (APF) have been around since 1986 and are extremely easy to implement, but they pose several theoretical limitations/shortcomings. The APF method has become a common tool in mobile robot applications in spite of these limitations. [2]

The limitations of the APF method come in three major categories. First, the existence of local minima not corresponding to the goal. Second, oscillations and instability when moving through a narrow space between two obstacles or a doorway. Third, the target is unreachable when located too close to an obstacle due to the repulsive obstacle potential overwhelming the attractive goal potential.

For discussion purposes, the Additive Attractive/Repulsive Potential Field method will be used here for its simplicity [3]. In this method the goal attracts the vehicle while the obstacles repel it. The potential function is simply constructed as the sum of the attractive and repulsive potentials. The attractive potential monotonically increases with distance from the goal. By following the negated gradient (gradient descent), the vehicle will trace a path to the goal. The repulsive potential keeps the vehicle away from the obstacles. The strength of the repulsive force depends on the vehicle's proximity to a given obstacle. The gradient is viewed as forces acting on a positively charged particle (the vehicle) which is attracted to the negatively charged goal. Obstacles have a positive charge so as to repel the vehicle. The APF can be viewed as a landscape where the vehicle moves "downhill" from a "high" value state to a "low" value state, stopping when it reaches a point where the gradient vanishes. Figure 10 shows how the APF path planner would perform in the benchmark case.

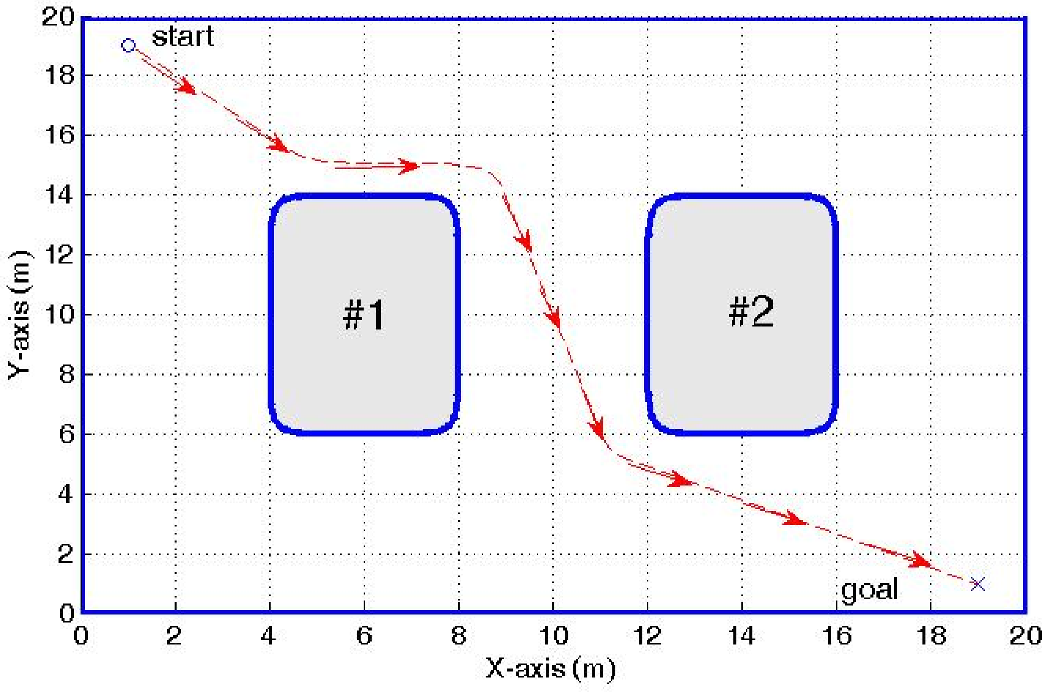


Figure 10. APF Planner in the Benchmark Case.

Much research has been conducted in developing the many variations and improvements of the APF method, most of which have been centered around improving the method's behavior in dealing with local minima that do not correspond to the target location. Figure 11 shows a very simple example of a vehicle using the APF method getting trapped in a local minimum. Two approaches to solving the local minima problem are discussed in [3].

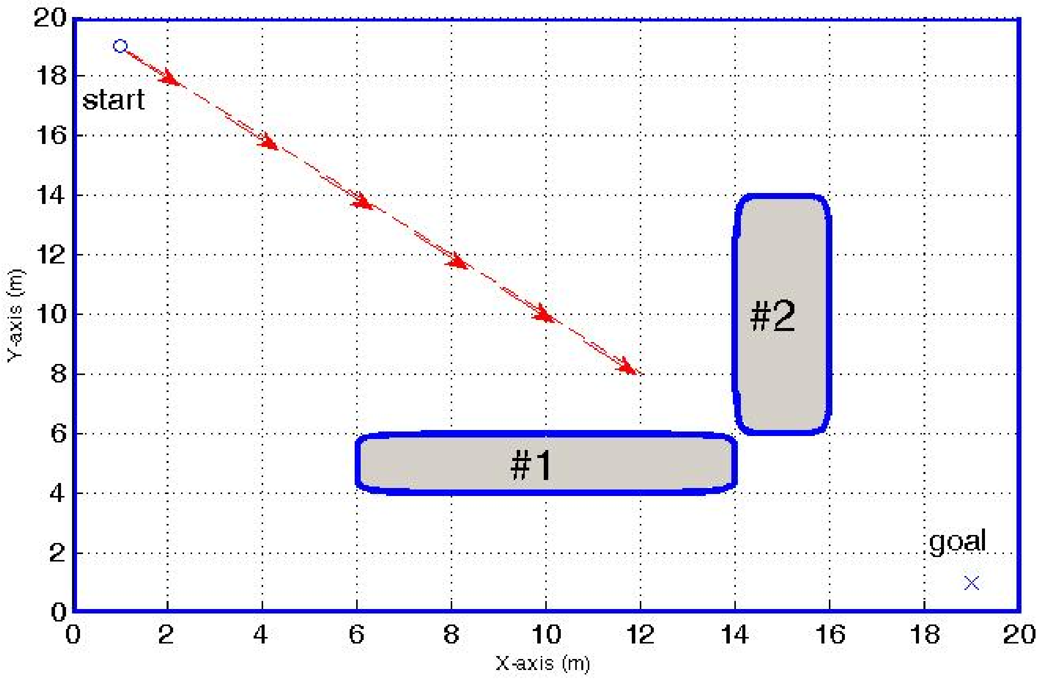


Figure 11. Local Minimum Problem in the APF method.

The first method, in [3], dealing with the local minima problem is to augment the potential field with a search-based planner. The Randomized Path Planner (RPP) is an example of such an approach. RPP follows the negative gradient of the specified potential function and when stuck at a local minimum, it initiates a series of random walks; that is, a series of random motions to escape the local minimum. RPP incrementally builds a graph of local minima, where the path joining two local minima is obtained by combining the random motions with the gradient descent motions. RPP will find a solution, if one exists, and is a good method for determining feasible solutions. RPP is one of the first widely used sampling-based algorithms. Sampling-based methods employ a variety of strategies for generating samples (collision free trajectories of the vehicle). They are capable of dealing with vehicles with many degrees of freedom and with many different constraints, such as kinematic and dynamic constraints.

The second method, in [3], dealing with the local minima problem is to define a potential function with only one local minimum. The simplest such function is represented by the Wave-Front Planner, which can only be implemented in spaces that can be broken down as grids. Free space pixels are assigned a '0'. Any pixel that is part of an obstacle is assigned a '1'. The target pixel is labeled with a '2'. All '0' pixels neighboring the '2' (goal) are labeled with a '3'. All '0' pixels adjacent to '3' are assigned a '4'; and so on, expanding out from the goal until the start location is reached. Each pixel assigned a certain number is equidistant from the target point as any other pixel assigned that same number. The planner then determines the path via gradient descent. Construction of the wave front guarantees there will be a path to the goal, if one exists. The result is a potential function with one minimum (the goal). Figure 12 shows how the use of the Wave-Front Planner can solve the local minimum problem of Figure 11.

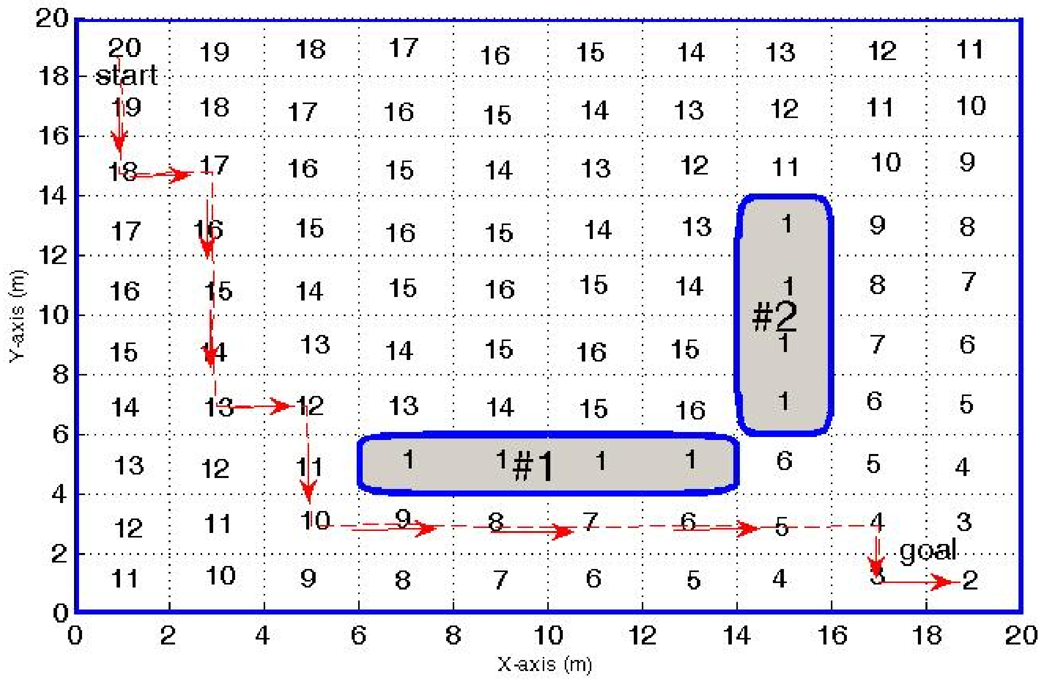


Figure 12. Wave-front Planner Overcomes Local Minimum Problem.

Some other solutions to the local minima problem are presented here. The local minimum problem is solved, in [8], by introducing an escape force along with the attractive and repulsive forces. Repulsive potential functions (RPF's) with angle distributions are introduced in [9]. The magnitude of the RPF's vary based on the angle between vehicle-to-goal and vehicle-to-obstacle. The simulated annealing algorithm (originally developed to simulate the anneal process of metal) is applied, in [10], to the APF method in both local and global path planning. A Coordinating Potential Field (CPF) method is used in [11]. This method uses a potential field function with tunable parameters and an adaptive genetic algorithm to optimize those parameters. All of the above methods, and many more, have shown success (to varying degrees) in solving the local minima problem.

The other two major limitations of the APF method are of lesser concern than the local minima problem, because in most cases the same approaches being used to overcome the local minima problem result in solving the other limitations as well. The RPF with angle distributions [9] overcome the oscillations and instability when moving through a narrow space and the unreachability of the target when located too close to an obstacle. The simulated annealing algorithm [10] and the CPF method [11] also perform successfully in overcoming those limitations.

- Path Planning and Control. The path to the target is solved, but not the controls. The control solution could be determined with additional programming modifications. The other option would be the addition of a trajectory following subsystem. Grade: C.

- Optimality. The APF method, by itself, is an efficient way of identifying safe paths for vehicles. However, although in some cases the results may be near optimal, there is nothing within the construct of the method to guarantee optimality. The use of a local-global (hybrid) planning strategy is demonstrated in [12]. It is optimal in the sense of globally minimizing the distance to the goal as well as locally minimizing the probability of hitting obstacles on the way. The technique has been used to control a Nomadics 200 robot. The local path planner is based on a potential field method using harmonic functions (solving Laplace's equation). The global path planner uses a search algorithm (Dijkstra's algorithm) on a graph structure representation of the map. The integration of the APF method with a global path planner was also demonstrated in [13]. The global planner (A-star algorithm) provides the minimum distance path through the known obstacles, and the APF method is then used to avoid the unknown obstacles as the vehicle traverses the path. A time-optimal behavior is provided by a wait or go strategy which determines if it is better to wait for obstacles to move out of the way or to keep moving to avoid them. The Evolutionary Artificial Potential Field (EAPF) [8] is run in real-time for path planning. It combines the APF method with genetic algorithms to derive optimal potential field functions. The APF's and cost functions have been designed with tunable parameters. These parameters are optimized by the multi-objective evolutionary algorithm (MOEA). MOEA is a stochastic search technique. This method furnishes the user the ability to decide what to optimize. The CPF method [11] is very similar to the EAPF method and therefore also can derive near optimal results. Grade: C.

- Obstacle Avoidance. Static obstacle avoidance is one of the APF method's strong points due to its ease of implementation. It has already been stated that by itself, it is an efficient way of identifying safe paths for vehicles. Many different hybrid path planning methods have been developed ([12] and [13] to name a couple) which use the APF method to avoid obstacles along the way as the vehicle travels along the path previously established by a global planner. A simple modification is needed to extend the method for use in dynamic obstacle avoidance. It needs only to process the entire algorithm at each environment alteration. In the case of a moving obstacle (where the environment is continuously changing) the algorithm must evaluate the potential field at each time instant. If the course and speed of an obstacle is known, that data can be incorporated into the potential field evaluation as well. Grade: A.

- Handling Vehicle Constraints. Vehicle kinematics/dynamics and nonholonomic constraints are not accounted for in this method. As in the case of Bug Algorithms, additional techniques are required in order to accomplish this task. The vehicle may not be able to move in the direction of steepest descent due to its orientation and constraints. The planner used in [12] uses a separate behavioral controller to map vehicle constraints to the planned trajectory. It works with the path planner to determine feasible trajectories. A feedback controller for a two-wheeled mobile robot is proposed, in [14], and successfully directs the robot to the goal from varying starting configurations without collisions with obstacles. Grade: C.

- Global vs. Local Information. The APF method can use both local and global information. It can plan a trajectory using global information; it can be used simply with local sensory information to conduct obstacle avoidance; or it can perform both tasks simultaneously. It can be used in an information centric way. It can continually account for the *big picture* by updating its map and evaluating the potential field at each time instant. Grade: A.

- Computational Complexity. This method can be used in real-time to solve the vehicle trajectory at each time instant or environment alteration when used for local obstacle avoidance or for smaller global problems. However, as the complexity and size of the problem grows (as is the case in real world missions), there is no guarantee that this method can be quick enough for real-time operation. Grade: C.

- Portability. This behavior implies changes to other behaviors already discussed. For example, the APF method can handle changing obstacle environments as it is well designed for both static and dynamic obstacle avoidance. However, a totally different dynamical system will have different constraints, and the APF method has already been evaluated as needing additional techniques or subsystems to handle constraints. As another example, it cannot be asked to follow a certain path (outside the steepest descent path) without some form of trajectory following subsystem. Grade: C.

- Completeness. A solution is guaranteed, if one exists (Wave-Front Planner). Grade: A.

- Multiple Vehicles. Path planning using the APF method for multiple vehicles is not common in the literature. Various methods have been developed utilizing decoupled planning, control and prioritization [15, 16, 17]. Centralized control techniques for the path planning and obstacle avoidance of vehicles in specific formations have been developed [18, 19, 20, 21]. These techniques and ideas could be extended for the planning of vehicles not in specific formations. Grade: A.

- Handling Errors and Uncertainties. Success here assumes the method can be used in real-time to solve the vehicle trajectory at each time instant or environment alteration. By using the method in real-time, the vehicle continuously updates the path to account for its positional error off the previous trajectory to prevent these errors from blowing up as the vehicle moves along in time. Grade: A.

3. Roadmaps

Roadmap Path Planning is the method of choice (almost exclusively) by all the semi-finalists of the DARPA Urban Challenge, which is an annual event requiring teams to build an autonomous vehicle capable of driving in traffic and performing complex maneuvers such as merging, passing, parking and negotiating intersections on its way from a start position to some goal location. The high level planning of the participant vehicles is done using Roadmaps and graph search algorithms such as the A* algorithm or Dijkstra's algorithm. Reactive planning (i.e., avoiding obstacles and traffic) is done using a variety of other path planners or obstacle avoidance algorithms, some of which have already been discussed in this work.

Roadmaps are a connected set of Nodes and Edges. A Node corresponds to a specific location, and an edge corresponds to a path between neighboring locations (Nodes). A vehicle uses the Roadmap like we use highway systems. The bulk of the motion occurs on the highway system (Roadmap) to get from "near" the start to "near" the goal. [3] To put it another way, a Roadmap is a network of road (path) segments for robot motion planning (vehicle path planning in this work). With a Roadmap, path planning is reduced to connecting the initial and goal positions of the vehicle to the road network, then searching for a series of roads from the initial vehicle position to its goal. The challenge is to construct a set of roads that together captures the connectivity of the environment (enabling the vehicle to go anywhere), while minimizing the number of total roads. [2] This description would imply that Roadmaps require global information. This is only true in that the initial Roadmap is constructed with all the information that is known *a priori*. It can then be modified and grow as the vehicle learns more about the environment. Typically, a graph search algorithm, such as Dijkstra's algorithm or the A* algorithm, is used to determine the shortest path from start to goal on the Roadmap.

We start with the two most common types of discrete Roadmaps; Visibility Graphs and Voronoi Diagrams. Both types maintain a mapping of the entire environment. As the complexity of the environment increases (i.e., 3D versus 2D, higher degrees of freedom, more obstacles, etc.) the use of these types of Roadmaps becomes computationally intractable. For this reason, very little can be found in the literature involving path planning using these types of Roadmaps.

In the Visibility Graph, Nodes are comprised of the initial point, the target point, and the vertices of the obstacles (which must be described by polygon shapes). All Nodes which are visible from each other are connected by straight-line segments. The Roadmap is made up of these segments along with the edges of the obstacle polygons. The task of the path planner is to find the shortest path from the initial position to the target along the roads defined by the Visibility Graph. [2] Figure 13 shows the Visibility Graph for the benchmark case. The dotted lines and the obstacle edges make up the roads. The arrows show the shortest path solution from start to goal.

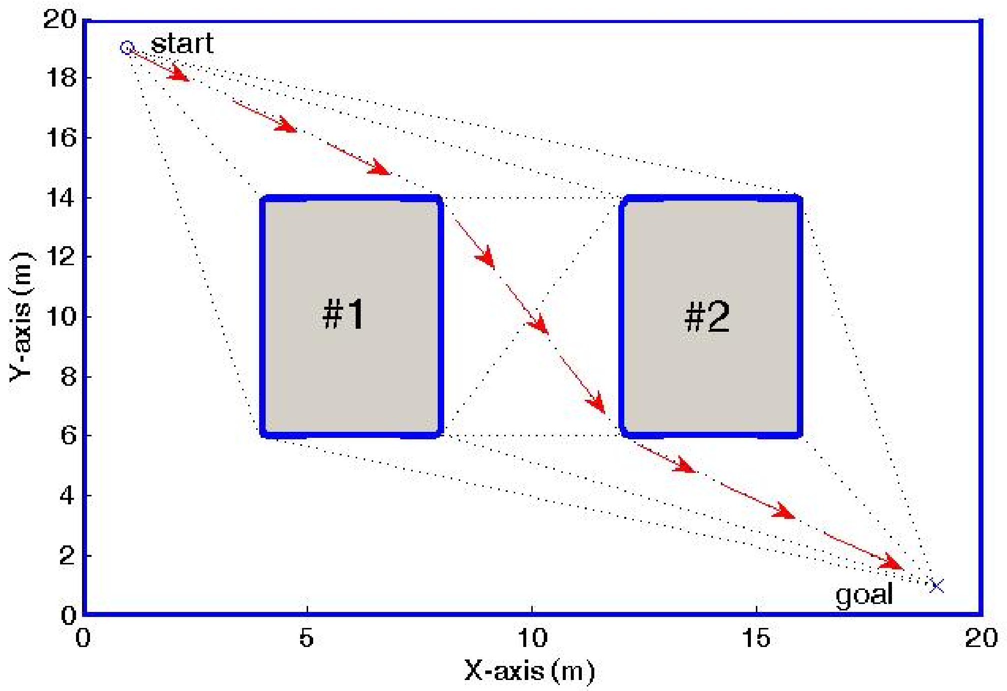


Figure 13. Visibility Graph for the Benchmark Case.

Visibility Graphs are popular in mobile robotics, partly because implementation is quite simple. They are extremely fast and efficient in sparse environments, but can be slow and inefficient compared to other techniques when used in densely populated environments. The solution paths are optimal in terms of path length, but the vehicle comes dangerously close to obstacles. [2]

The Voronoi Diagram consists of the lines constructed from all points that are equidistant from two or more obstacles. The planner must first connect the start point and the target location to the Roadmap. The task of the path planner is then to find the shortest path from the initial position to the target along the roads defined by the Voronoi Diagram. Figure 14 shows the Voronoi Diagram for the benchmark case. The dotted lines make up the roads. The arrows show the shortest path solution from start to goal. Like the Visibility Graph, the Voronoi Diagram is simple to implement but becomes slow and inefficient in densely populated environments. The solution paths provide safe distance from obstacles, but this results in a non-optimal (in terms of path length) solution and can pose problems for short-range sensors seeing the surrounding obstacles.

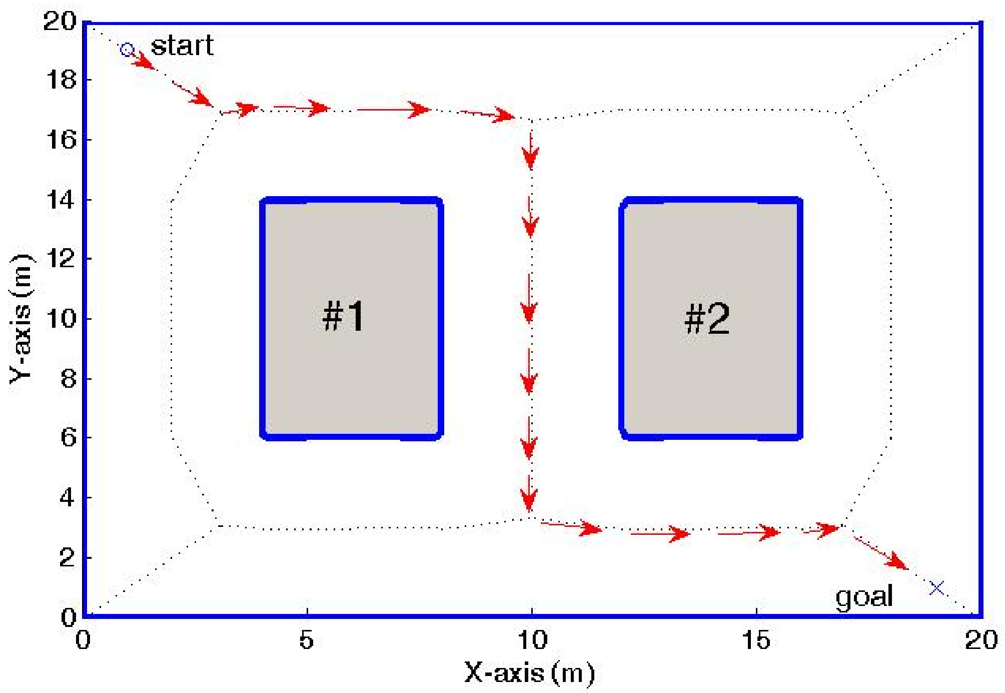


Figure 14. Voronoi Diagram for the Benchmark Case.

It has already been stated that as the dimensions and complexity of the environment increases, the discrete Roadmap planners discussed thus far become slow and inefficient. The problem with those discrete planners is they require an explicit representation of the environment and its geometry. As was done in the APF method, we turn to sampling-based planners to improve on our Roadmap theory. These types of planners are described in detail in [3]. Sampling-based planners do not explicitly construct obstacle boundaries, but rely on a procedure that can decide whether a given vehicle configuration is in collision with the obstacles or not.

The Probabilistic Roadmap (PRM) planner [3] has its beginnings in the early to mid 1990s. It is easy to implement and works well even with rather high-dimensional and complex problems. It is divided into 2 phases (Learning and Query). In the Learning Phase the basic PRM planner uses a uniform random distribution to conduct Node sampling and thus builds the Roadmap in a probabilistic way. This is just one of many sampling schemes in the literature. The Edges are collision free paths between Nodes computed by a simple and fast local planner. This local planner can be as simple as connecting two Nodes with a straight line. Given that a simple and fast local planner is desired, there must be a fairly dense distribution of Nodes to ensure successful Node connections. A more powerful local planner would allow the use of fewer Nodes, but would cause the planner to be too slow. So to reiterate, the preferred PRM planner uses a very fast local planner with a large, dense Roadmap. Roadmap construction is done by first finding the *n* Nodes. For each Node *q*, the local planner attempts to connect each of the *k* closest neighbor Nodes to *q*. When an attempt is successful, the path is added to the Roadmap. In the Query Phase, user-defined initial and target points (vehicle configurations) are connected with the pre-computed Roadmap using a simple and fast local planner in the same way that Nodes are connected to each other, and the Roadmap is used to solve the path planning problem. As in the previous Roadmap examples, a graph search algorithm, such as Dijkstra's algorithm or the A* algorithm, is used to determine the shortest path from start to goal on the Roadmap. The basic PRM planner was conceived as a multiple-query planner, but can be used in single queries; however, it is not the fastest planner for single queries. Other single-query planners exist for that purpose. Figure 15 shows what a PRM solution might look like in the benchmark case. For each Node *q*, each of the three closest neighbor Nodes were connected. The two main disadvantages to the PRM planner are that they are less efficient at determining paths through narrow passages (or closely spaced obstacles), and although they are faster than the more primitive Roadmap planners, they are still too slow to be operated in real-time. There has been much work conducted and documented in the literature on improving on these drawbacks. Choosing different sampling strategies (vice just a uniform random distribution) and connection strategies (besides the *k* closest neighbors) have improved on the narrow passage problem. Creating sparse Roadmaps by not allowing cycles to be created can speed up Roadmap construction. The paragraphs that follow cover some of the proposed solutions to the narrow passage and run time problems.

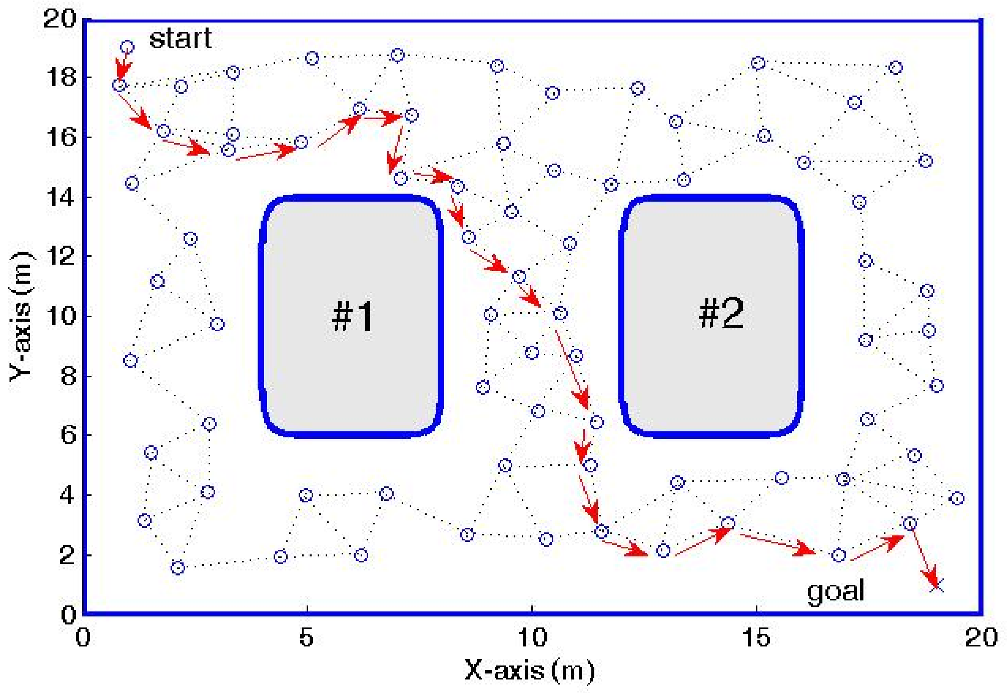


Figure 15. PRM Planner for the Benchmark Case.

Single-query sampling-based planners are described in detail in [3]. They attempt to solve a query as fast as possible and do not focus on the exploration of the entire free space. Expansive-Spaces Trees (ESTs) and Rapidly-Exploring Random Trees (RRTs) are singe-query path planners. They are very efficient for kinodynamic (taking kinematic and dynamic constraints into account) path planning. They bias the Node sampling by maintaining two trees that branch out from the initial point and the target point. These trees are grown toward each other until they are merged into one, at which point a solution is returned. These methods have improved the speed of Roadmap planning considerably.

The Sampling-Based Roadmap of Trees (SRT) planner is described in [3]. It effectively combines the basic PRM planner with tree methods such as ESTs or RRTs. It replaces the local planner of the PRM method with a single-query sampling-based tree planner, like EST or RRT, enabling it to solve problems that other planners could not. The SRT planner can be used as a multi or single query planner. It constructs a Roadmap as in PRM, but the Nodes of the Roadmap are trees, constructed by the tree planner. In other words, SRT constructs a Roadmap of trees. SRT's speed comes from the fact that the trees can be constructed independently of one another, allowing for an efficient parallel processing. It can be used to solve very high dimensional problems with kinematic and dynamic constraints.

It is shown in [22] that significant, scalable speedups can be obtained for path planning by parallelizing both the Node generation and Node connection processes of the PRM method.

A randomized motion planner for kinodynamic asteroid avoidance problems is presented in [23]. A robot must avoid collisions with moving obstacles under kinematic and dynamic constraints, and reach a specified goal state. The planner samples the state-time space of the robot by picking control inputs at random in order to compute a Roadmap that captures the connectivity of the space. However, the planner does not precompute a Roadmap as most PRM planners do. Instead, for each planning query, it generates, on the fly, a small Roadmap that connects the given initial and goal state. In contrast to the basic PRM planners, the computed Roadmap is a directed graph oriented along the time axis of the space. The efficiency (and speed) of the planner makes it possible for a robot to respond to a changing environment without knowing the motion of moving obstacles *a priori*.

The SBL (Single-query, Bi-directional, Lazy collision checking) planner [24] is a PRM planner that combines a single-query bi-directional sampling strategy with a lazy collision-checking connection strategy. The planner postpones collision tests until they are absolutely necessary. Significant improvements in efficiency and speed were shown using this planner. Experiments showed that SBL is between 4 and 40 times faster than classical single-query bi-directional PRM planners.

The RRF (Reconfigurable Random Forest) planner [25] can learn incrementally on every planning query and effectively manage the learned Roadmap as the process goes on. RRF is a modified version of the RRT method. It can account for environmental changes while keeping the size of the Roadmap small. The planner removes invalid nodes in the Roadmap as the obstacle configuration changes. It also uses a tree-pruning algorithm to trim RRF into a more concise representation. Experiments show the planner's flexibility, efficiency and speed.

The RET (Rapidly Exploring Evolutionary Tree) planner [26] is a hybrid RRT planner employing an online Evolutionary Algorithm (EA) to formulate additional biases (solution guesses) prior to each recalculation of the trajectory. An offline EA is utilized to produce a bias in an obstacle filled environment prior to rearranging the obstacles. This bias reflects the original environment and improves the RRT's efficiency during replanning in environments with moving obstacles. The algorithm is fast and allows the vehicle to continuously monitor the environment and recalculate planned trajectories.

A new approach for building and querying PRMs called Customizable PRM (or C-PRM) is presented in [27]. C-PRM improves both drawbacks of the PRM planner; it speeds up the process and solves the narrow passage problem. In the Roadmap construction stage, C-PRM builds coarse Roadmaps by performing only an approximate validation of the Roadmap nodes and/or edges. In the query stage, the Roadmap is validated and refined only in the area of interest for the query. This approach, which postpones some of the collision checks to the query phase (lazy collision checker, as in the SBL planner), has been shown to improve the efficiency and speed of PRM planners.

Most of the current literature on PRM methods is devoted to solving the narrow passage problem (or closely spaced obstacles). In addition to the C-PRM [27] already discussed, one other approach is described here. The HFPRM (Harmonic Function-based PRM) planner [28, 29] is made up of three phases. In phase one, the Laplace's equation, pertinent to potential flow, in an environment cluttered with obstacles is solved. In phase two, a PRM is constructed based on information obtained about the environment topology through the HF technique developed in phase one. This allows for the biasing of the Node selections. More random Nodes are generated in high velocity regions, resulting in better coverage in narrow passages. The Roadmap is then searched for the shortest path in phase three. Other approaches to the narrow passage problem can be found in the literature. Some notable examples can be found in [30, 31, 32, 33].

- Path Planning and Control. Similar to the APF method, a trajectory to the target is found, but not the control solution. The vehicle would require the addition of a trajectory following subsystem. The other issue here is the fact that each edge connecting any two Nodes is determined irrespective of how the surrounding edges connect to the same Nodes. This requires a post-processing algorithm to smooth out the path. In [34], a motion planning framework is presented for a fully deployed autonomous unmanned aerial vehicle which integrates the PRM and RRT methods. The framework was verified in actual flight using the WITAS RMAX helicopter. At the control level, the path is executed using a Dynamic Path Following (DPF) controller. Grade: C.

- Optimality. It was stated earlier that the Visibility Graph provides a solution that is optimal in terms of path length. The problem is that the method is computationally intractable in more complex environments with higher degrees of freedom. PRM planners can be designed to simultaneously optimize chosen vehicle performance parameters within the construct of the developed Roadmap. An example of this can be found in [35], which presents a method for extracting optimal paths from Roadmaps. The problem with PRM planners is that the optimization can only be as good as the generated Roadmap. A PRM planner cannot guarantee a minimum distance or minimum time trajectory, because it is designed to find a connection from start to goal, not an optimal connection. In Roadmap construction, once there is a complete path to the target, the construction stops; it does not continue looking for shorter paths. Planners can be designed to continue looking for better paths, but due to the random nature of PRM planners, the optimal path may never be found, and if it is found we have no way of knowing if it is actually a near optimal trajectory. Grade: F.

- Obstacle Avoidance. PRM methods can account for both static and dynamic obstacles. It has already been stated, in [23], that the efficiency (and speed) of the planner makes it possible for a robot to respond to a changing environment without knowing the motion of moving obstacles *a priori*. Because of the very short running time of the planner in [23], trajectories can be replanned at each time instant or detected environment alteration. The incremental Roadmap building algorithm, in [36], is able to effectively deal with the system's dynamics, in an environment characterized by moving obstacles. The RRF planner [25] removes invalid Nodes after an obstacle change and reconstructs the RRF structure, thus allowing it to manage the Roadmap effectively and efficiently as the environment changes. Other examples of dynamic replanning are presented in [26, 34], and successfully tested on the WITAS RMAX helicopter [34]. Grade: A.

- Handling Vehicle Constraints. Random sampling techniques perform well in path planning problems beyond just pure geometric path planning. PRM methods are capable of handling kinematic and dynamic constraints. They can be used to plan paths for nonholonomic vehicles. Much research has been conducted on this topic. For example, a Node need not just describe a location that is collision free, but may also describe the vehicle orientation as well. To this end, edges do not have to be limited to merely being straight line segments between Nodes, but can correspond to specific behaviors or motion commands that enable the vehicle to move from one Node to another. The planner in [23] demonstrates the ability to account for kinodynamic vehicle constraints and has been applied to nonholonomic vehicle navigation. C-PRM [27] allows the user to request a path that meets certain requirements/constraints. For example, one might ask for a path that is collision free, restricts certain vehicle parameters to a particular range (i.e., wheel turn angle), and has some particular clearance from obstacles. Invalid Nodes and Edges are removed as they are discovered, and the process iterates, resulting in a Roadmap that is customized with respect to the requirements. Other approaches that are successful in handling constraints can be found in the literature, and some examples are in the references [34, 36]. Grade: A.

- Global vs. Local Information. The PRM method can use both local and global information. It can plan a trajectory using global information, and then use local sensory data to dynamically replan the trajectory at each time instant. It can be used in an information centric way. It can continually account for the *big picture* by updating its map at each environment alteration. The RET planner [26] is a good example of the benefits of using all available information in producing a trajectory and how information can be used to speed up subsequent replanning. Grade: A.

- Computational Complexity. The improvements in probabilistic methods shown thus far in [22, 23, 24, 25, 26, 27, 28, 36] indicate that PRM-based methods can be used in real-time to replan the vehicle trajectory at each time instant or environment alteration. However, as the complexity and size of real world missions grow, there is no guarantee that this method can be quick enough for real-time operation. Grade: C.

- Portability. PRM-based methods can be adapted to new obstacle environments or even totally different dynamical systems. This method can be used in three dimensions, and it has been shown as being capable of accommodating many degrees of freedom. For example, the HFPRM planner [28, 29] can be applied to virtually any type of robot. As in the APF method, however, a PRM planner cannot be asked to follow a certain path (such as a zigzag pattern) without some form of trajectory following subsystem. Grade: C.

- Completeness. In the case of explicit Roadmaps (which map out the entire environment) like the Visibility Graph or the Voronoi Diagram, a solution is guaranteed if one exists. In the case of PRM planning methods, the issue of completeness must be defined a bit further. If the sampling is deterministic, including quasirandom or sampling on a grid, then the planner is said to be resolution complete [3]. If the sampling is random, then this form of completeness is called probabilistic completeness, because as time goes to infinite the planner will successfully find a solution if one exists [3]. Grade: A.

- Multiple Vehicles. It has already been stated that if the planner can handle moving obstacles (as this one can), then planning paths for multiple vehicles in a decentralized (decoupled) manner is automatically possible by treating other vehicles as moving obstacles. Optimal Motion Planning for Multiple Robots is presented, in [37], by defining a state space that simultaneously represents the configurations of all of the robots. The SBL planner [24] has been proven to be more reliable at centralized planning than decoupled planning. Experimental validation was achieved on a spot-welding station with 2 to 6 robot manipulators combining 12 to 36 degrees of freedom. Grade: A.

- Handling Errors and Uncertainties. Success is achieved under the assumption that this method can be used in real-time to replan the vehicle trajectory at each time instant or environment alteration. By using the method in real-time, the vehicle can continuously update its position and provide this new position update to the next replanning iteration. This accounts for its positional error off the previous trajectory to prevent these errors from blowing up as the vehicle moves along in time. Grade: A.

4. Cell Decomposition

Cell Decomposition is a purely geometric method whose idea is to discriminate between cells (geometric areas) that are free and cells that are occupied by obstacles or objects. [2] The basic procedure as described in [2] is as follows: 1) Divide the environment into connected regions called "cells. " Much of the literature is devoted to how the environment is decomposed into cells. If the boundaries of the cells are determined as a function of the structure of the environment, such that the union of the cells is equal to the free space, then the method is "Exact." If the decomposition results in an approximation of the actual map, then it is termed "Approximate." 2) Determine which cells are adjacent and construct a "connectivity graph." Two cells are adjacent if they share a common boundary. The connectivity graph is made up of Nodes (corresponding to cells) and Edges (corresponding to the adjacency of the Nodes). 3) Determine the cells that contain the start and goal points. 4) Search for a path in the graph to join the cells containing the start and goal positions. This is done using graph search techniques such as the A* or Dijkstra's algorithms. 5) Compute a path within each cell using a simple local planner, such as a straight line segment through the midpoints of the cell boundaries.

Exact Cell Decomposition (ECD) uses the geometry of the environment to determine the size and shape of the cells. The resulting cells are either completely free or completely occupied, resulting in a complete path planner. The position of the vehicle within each cell of free space does not matter; what matters is its ability to traverse between adjacent cells [2]. The most popular Exact method is the Trapezoidal Decomposition [3], which is based on 2D cells shaped like trapezoids or triangles (which are trapezoids with one side having a zero length). All obstacles are polygonal shapes, and cell boundaries are formed by drawing vertical extensions above/below each vertex of every obstacle. Figure 16 shows the ECD method in the benchmark case. Dotted lines show the boundaries of the individual cells. The local planner constructs the path by connecting the midpoints of each vertical extension, yielding a collision-free path through the free space. This is a very efficient path planner in sparse environments (resulting in a low number of cells), however, it becomes computationally intractable when the environment becomes more and more complex (such as higher degrees of freedom, denser obstacle environments, etc.), meaning that it will take too much time to solve for a trajectory to reach the goal, making this method unusable in real-time applications. The fundamental disadvantage here is the need to have an exact model of the entire environment. As a matter of fact, this method is rarely used in mobile robotics applications. A better way to go would be to discretize the environment without the need for an exact mapping of all the obstacles.

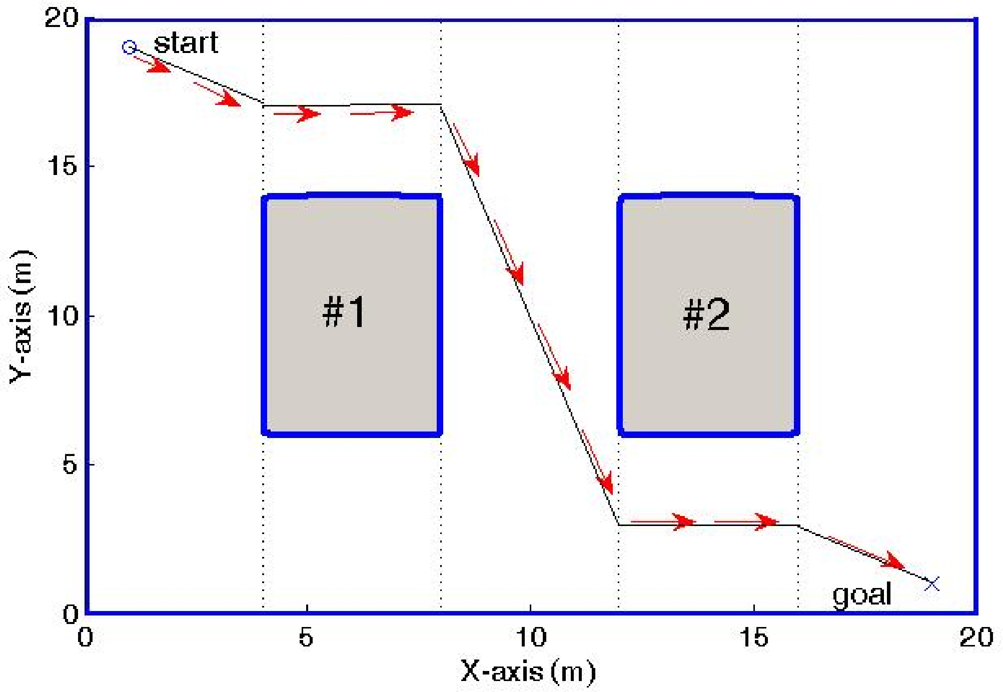


Figure 16. ECD for the Benchmark Case.

Approximate Cell Decomposition (ACD), in contrast to the Exact method, is one of the most popular techniques for mobile robot path planning due to the popularity of grid-based environmental representations. This is the single most common map representation technique currently utilized (outside of sampling based probabilistic methods). [2] We start with fixed-size cell decomposition, as described in [2]. Cell size is not dependent on the obstacle environment, but must be small enough to capture the resolution of narrow passages. The obvious benefit is the low computational complexity, because the entire obstacle environment need not be modeled. The grid is generated using a fixed cell size and shape. All empty cells become Nodes in the Roadmap, while any cell obstructed by an obstacle is removed. The cost of this method is the memory needed to store the grid and the time it takes to verify empty cells. Even in a large, sparse environment, the grid must be represented in its entirety. Figure 17 shows the fixed-size ACD method in the benchmark case. Dotted lines show the boundaries of the individual cells. It is purely coincidental (due to the nice size, shape and positioning of the obstacles) that the union of the cells in this case covers all of the free space. The Cye Personal Robot is an example of a commercially available robot that performs all its path planning on a 2D 2cm fixed-cell decomposition of the environment. [2] It is designed to avoid known (static) obstacles and travels along known routes.

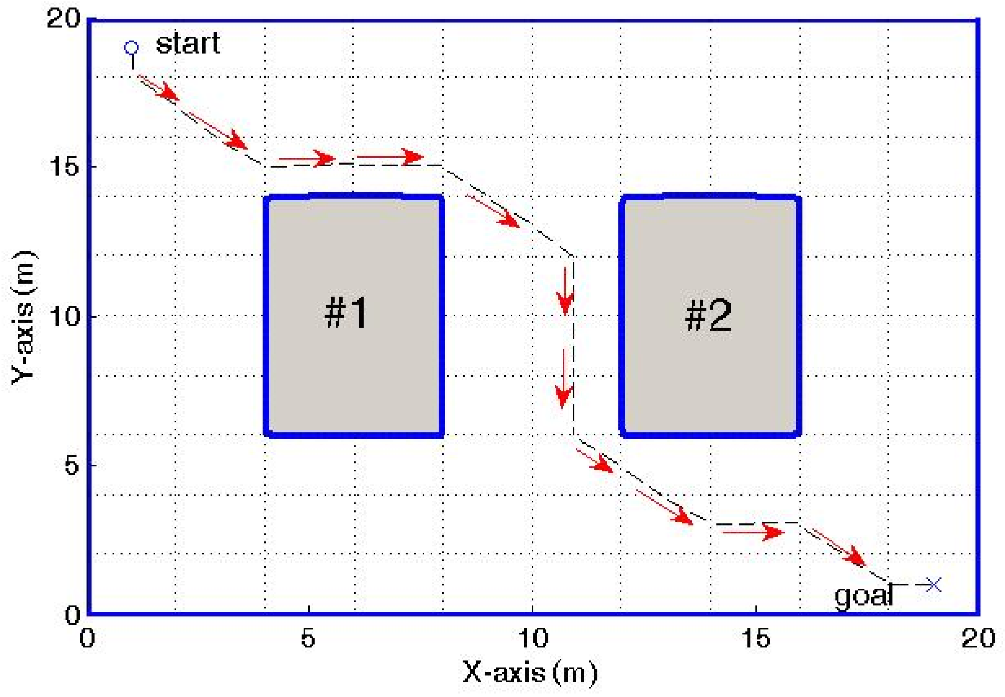


Figure 17. Fixed-size ACD for the Benchmark Case.

A better ACD method is the variable-size method, which is also known as a multi-resolution or hierarchical method [38, 39, 40]. It is less complex and easier to implement than the fixed-size decomposition. It does not have the memory cost of the fixed-size method because sparse environments will have larger cells (and thus fewer cells). As described in [2], this method starts with equal size rectangles. Cells that lie entirely inside an obstacle are classified as full cells and are discarded. Only cells whose interiors lie entirely in the free space (empty cells) are used to construct the connectivity graph. In the Quadtree method, if a cell is found to be in collision with an obstacle (mixed cell), that cell is further decomposed into four identical new rectangles. The resolution is reduced until either a path is found or a resolution limit is reached. This method is said to be resolution complete. Figure 18 shows the variable-size ACD method in the benchmark case. Dotted lines outline the progression of the decomposition, and solid lines outline the free cells included in the Roadmap.

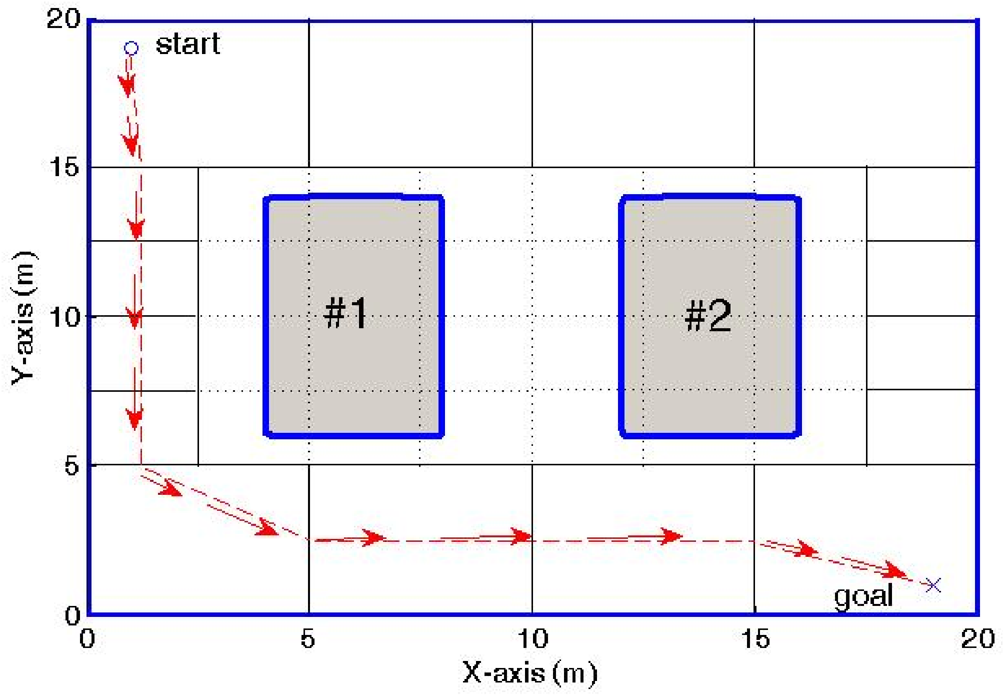


Figure 18. Variable-size ACD for the Benchmark case.

Cell Decomposition methods are the most common and widely used in path planning applications because of their simplicity. By itself, the Cell Decomposition method is the most popular method for path planning in a static known environment as an offline planner from which path queries can be made. However, its use in real-time applications becomes limited by the fact that the complexity of the Roadmap grows exponentially with the number of degrees of freedom. It harbors the same speed disadvantage as previous methods requiring the search of the entire free space to produce a solution. Besides offline path planning in static obstacle environments, it also owes its popularity to its ability to make other path planners better and its use in hybrid applications. In [41], Cell Decomposition and the APF method are used together to form a hybrid path planner. Global planning is done offline using Cell Decomposition, then online local planning is achieved using the APF method to account for unknown and dynamic obstacles. In [42, 43], ACD is used to bias the probabilistic sampling of the PRM planner. This helps to speed up the construction of Roadmaps and solves the narrow passage problem that plagues the basic PRM method. In [44], Cell Decomposition is applied to both the APF method and sampling-based methods. Cell Decomposition methods are also the method of choice for other useful applications. Morse Cell Decompositions can be used to achieve coverage of an entire area, such as a robot that vacuums a room. [3] Visibility-based Decompositions are used to address the pursuit/evasion problem. [3]

- Path Planning and Control. The path to the target is solved, but not the controls. The path that is obtained is piecewise linear, therefore some post-processing must be performed in order to smooth out the path. A trajectory following subsystem is needed to follow the path. Grade: C.

- Optimality. Using a high enough resolution, Cell Decomposition can provide a minimum distance solution. But as already discussed, the higher the resolution the slower the algorithm. Furthermore, since this method does not account for any constraints other than spatial/geometric, other techniques that can handle kinodynamic constraints would be required in order to optimize other parameters. Grade: C.

- Obstacle Avoidance. It has already been stated that the Cell Decomposition method is the most popular method for path planning in a static known environment as an offline planner. The speed of this planner is not sufficient to run it in real time in order to continuously replan in a changing environment. Work has been documented in the literature on improving the speed and efficiency of this method. In [39] it is proposed that this method lends itself to parallelization, thus allowing the use of multiple processors in order to improve its speed and performance. In [45], the incremental algorithm concept is introduced in which the initial problem is solved offline, and in real-time the planner needs to merely update the current mapping rather than completely replan. The idea here is that the environment only undergoes an incremental change from one time instant to the next. Although proposed here as a technique, it was not applied to moving obstacles. A new approach to path planning in high-dimensional static environments was presented in [46, 47] called Probabilistic Cell Decomposition (PCD). PCD combines Cell Decomposition with probabilistic sampling. This technique prevents the need for searching the entirety of each cell to determine if it is empty, full or mixed. It saves time by simply assuming all cells to be possibly free until proven otherwise. As a path is being generated, if the local planner discovers a collision in a cell, the cell is further sampled to determine if it is full or mixed. If full, it is discarded from the Roadmap. If mixed, it is further decomposed. The PCD planner was experimentally tested on a Puma 560 robot arm mounted on a Nomadic XR4000 mobile platform. [47] PCD's speed and efficiency, although an improvement over other Cell Decomposition methods, has not yet been shown as sufficient enough for real-time use or dynamic obstacle avoidance. To date, the Cell Decomposition method is still primarily being used as the offline planner while other methods such as APF are being used for obstacle avoidance. Grade: C.

- Handling Vehicle Constraints. Vehicle kinodynamic and nonholonomic constraints are not accounted for in this method. Additional techniques would be required in order to accomplish this task, such as adapting the vehicle controller to the Roadmap and throwing out infeasible paths. Grade: C.

- Global vs. Local Information. Going back to the obstacle avoidance argument, to date, the Cell Decomposition method is still primarily being used as the offline planner while other methods such as APF are being used to handle local information. When combined with other methods it produces a good hybrid strategy for handling global and local information. Grade: C.

- Computational Complexity. The arguments presented in obstacle avoidance indicate this method does not lend itself to real-time operation. The first and only work found in the literature that indicates the possibility of real-time performance is [38], which states that a hierarchical multiresolution cell decomposition is suitable for online implementation. It presents an ACD scheme based on the Haar wavelet transform.

It shows improvements in the use of computer memory and speed, but does not actually perform any real-time simulations. Grade: F.

- Portability. Since this method does not take into account the kinodynamics of the vehicle, as a static obstacle offline path planner, it can be used in varying environments and applications. However, as the complexity of the environment increases, such as increasing the degrees of freedom, this method becomes much slower. As in the other methods, this method cannot be asked to follow a certain path (such as a zigzag pattern) without some form of trajectory following subsystem. Grade: C.

- Completeness. This method is resolution complete. It may require a high resolution to find a solution, and thus take a long time to solve, but if a solution exists it will find it. The PCD planner is probabilistically complete. Grade: A.

- Multiple Vehicles. Based on the method's need for help in avoiding moving obstacles, it would similarly need help in avoiding other vehicles. Very little could be found in the literature involving path planning for multiple vehicles using this technique [40], however it is assumed that with the help of other techniques, multi-vehicle path planning could be accomplished. Grade: C.

- Handling Errors and Uncertainties. The vehicle must use feedback along with some trajectory following subsystem to reach its goal based on the original *offline* solution. Grade: C.

5. Optimal Control

Optimal Control Trajectory Planning with Numerical Optimization, as described in [3], is a direct approach to the complete motion-planning problem, which determines the path to the target by searching within the vehicle's state space. The result is the complete state space and control solution from start to goal. Numerical optimization is added as part of the technique, because these problems are too complex to be solved analytically. With a good numerical solver, the solution becomes a matter of good problem formulation. This means the variables must be well scaled and balanced, the dynamics and constraints must be clearly defined, all equations/functions (dynamics, cost, constraints) must be sufficiently smooth, and finally, a good initial guess is advantageous to bias the solution (to be called a bias from here on out). A bias is not always needed, but as the obstacle environment becomes more complex, a bias will help to prevent infeasible solutions from being generated. The use of a bias also serves to speed up solution times when the planner is used in real-time.

The basic concept of how Optimal path planning works is as follows from [3, 48]. First the planner must be given the kinodynamic equations of the vehicle, the obstacles to be coded into smooth path constraint functions, and the cost function. The kinodynamic equations can be viewed as constraints as well (like the obstacles), defining the relationship between the vehicle state and the control input. The obstacles need not be smooth, but the constraints used to define the obstacle must be made up of one or more smooth functions. The Optimal Control technique finds a solution to the state equations that takes the vehicle from the initial state at time zero to the final state at the final time, while avoiding obstacles and obeying vehicle state and control limits, and minimizing some cost function. The cost function can be any function of state variables, control variables and time, as long as it is sufficiently smooth (i.e., continuous and differentiable). Figure 19 shows the path planning solution for the benchmark case using the Optimal Control technique.

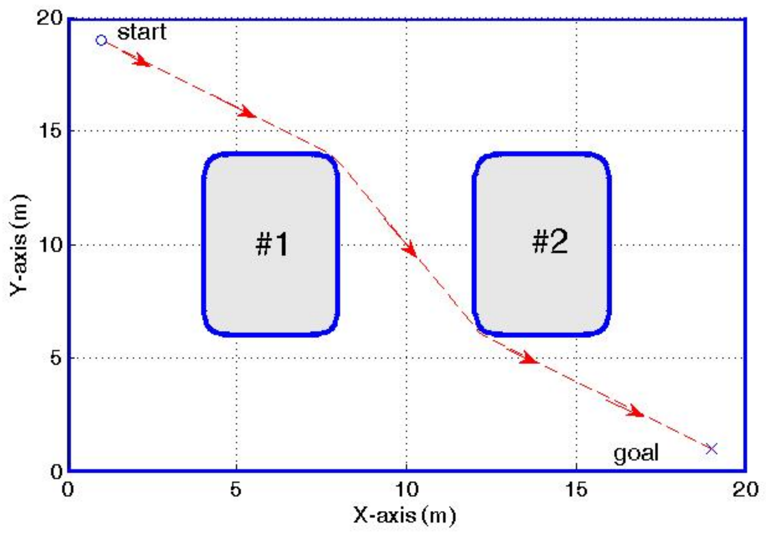


Figure 19. Optimal Control Technique in the Benchmark Case.

It is stated in [3] that there are two drawbacks to the Optimal Control technique. First, numerical methods require an initial bias to steer the solution in the right direction. Second, this method can be computationally costly and thus require excessively long solution times, at least from the standpoint of running the algorithm in real-time. Advances in Optimal Control and numerical optimization tools over the last five to ten years have made those drawbacks less and less of a problem in recent research.

We first address the bias that has traditionally been required in solving an Optimal Control problem numerically. The issue here is the fact that the need to provide a bias can take away from the autonomy of the problem; however this does not have to be the case. The initial offline run to determine a trajectory does not have to be autonomous. This first run is performed prior to sending the vehicle out on its mission. Future real-time runs must be autonomous, and by virtue of already having a solution from the offline run, the vehicle can use that solution as the bias for its real-time solutions. Subsequently, each real-time solution can be used as the bias for the next real-time run, thus preserving its autonomy. This new approach involves using the optimal control algorithm in a feedback form, thus self-generating the needed bias. Research has been done [1, 49, 50, 51, 52, 53, 54, 55] solving various path planning problems using Optimal Control with numerical optimization in a feedback control algorithm. The numerical optimization tool that is used, DIDO [56], incorporates pseudospectral (PS) methods with a sequential quadratic programmer, and therefore is capable of rapidly generating extremals for properly formulated Optimal Control problems. [1] Information and illustrations portraying DIDO's development, viability, and applicability for solving Optimal Control problems is found in [57, 58]. Many other numerical solvers exist, such as FSQP [59], NPSOL [60], and routines in the NAG [61] and MATLAB optimization toolbox libraries; however, DIDO [56] will be the numerical optimization tool of choice for the remainder of this work, due primarily to the fact that it is less sensitive to the need for a bias than the other methods. In [49], a novel time-optimal sampled-data feedback control algorithm is introduced for closed-loop control of NPSAT1 in the presence of disturbances. The feedback law is not analytically explicit; rather, it is obtained by a rapid re-computation of the open-loop time-optimal control at each update instant. The basic idea of this algorithm is to take the initial conditions and desired final states (as the initial bias), and conduct an offline run; the result is an initial control signal and an estimate of the final time. These are then used as the bias for the real-time closed-loop runs. Except for the initial input of start and goal states, the system is autonomous. In [50], an optimal nonlinear feedback guidance law was constructed for a reusable launch vehicle (RLV) based on the concept of using the offline open-loop solution to bias the first real-time closed loop solution, and each subsequent closed-loop run was biased by the previous solution. Again, autonomy was preserved. In [1, 51, 52, 53, 54, 55], the use of Optimal Control in a feedback form using a bias that is autonomously obtained is demonstrated on an RLV, a tricycle, a UGV, a UAV, and a slew problem for NPSAT1.

In reference to the computational cost and thus excessively long solution times, it has been shown, in [1, 49, 50, 51, 52, 53, 54, 55], that the use of a bias to help steer the trajectory solution speeds up the run times significantly from seconds to fractions of a second. So the bias does not just aid in achieving feasible solutions, but also speeds up the solution process considerably. Computation time can be reduced by at least a factor of 100 (a conservative estimate) by optimizing the actual code and eliminating the Windows and MATLAB overhead [50]. Also, advancements in sparse linear algebra, development of new algorithms, and improved computer processor speeds have made solving optimization problems relatively easy and fast [51]. Recent applications of real-time optimal control [1, 49-55] have proven to be very promising in facilitating feedback solutions to complex nonlinear systems [51].

One more interesting point is the connection that can be made between the APF method and the Optimal Control method by use of a form of optimal planning called Receding Horizon Planning. [62] The Receding Horizon Planner, as described in [62], is essentially an optimal planner except that the cost function is formulated over a small fixed time period instead of the entire time from start to finish. This type of planner has been used since the 1970s as a way to improve the speed of optimal control algorithms. As the planning horizon grows and reaches the final time, it becomes the Optimal Control planner. As the planning horizon shrinks to zero, it becomes identical to the APF method. So in this framework, the APF and Optimal Control methods are simply the two extremes of Receding Horizon planning. The noted speed improvements and recent applications of real-time Optimal Control have eliminated the need for Receding Horizon planning.

- Path Planning and Control. Optimal Control Path Planning with Numerical Optimization, as described in [3], is a direct approach to the complete motion-planning problem, which determines the path to the target by searching within the vehicle's state space. The result is the complete state space and control solution from start to goal. Grade: A.

- Optimality. The Optimal Control technique finds a solution to the state equations that takes the vehicle from the initial state at time zero to the final state at the final time, while avoiding obstacles and obeying vehicle state and control limits, and minimizing some cost function. The cost function can be any function of state variables, control variables and time, as long as it is sufficiently smooth (i.e., continuous and differentiable). Grade: A.

- Obstacle Avoidance. This is inherent in the solution to the Optimal Control problem, because obstacles are coded into smooth path constraint functions which must be satisfied as part of the solution. In [1, 52, 53, 54] this method's utility for static obstacle avoidance is well demonstrated. The dynamics of moving obstacles are coded into the constraint equations in [1, 52] to show the utility of using this method for dynamic obstacle avoidance when the future positions of obstacles can be predicted. It has already been shown that the speed of this method lends itself to real-time closed-loop feedback control. This allows for the algorithm to be processed at each time instant or environment alteration, which makes it viable for use with dynamic obstacles whose future positions cannot be predicted, or pop-up obstacles not known at the start of the mission. The ability to find a safe path around pop-up obstacles has already been demonstrated in [1, 52, 53]. Grade: A.

- Handling Vehicle Constraints. All vehicle and obstacle constraints are accounted for in the problem formulation. This is one of the many reasons why good problem formulation is paramount to the success of this method. Grade: A.

- Global vs. Local Information. The Optimal Control technique is a completely information centric path planner. It uses all the information that is known about the vehicle, the environment, and the parameters to be optimized. It initially plans a path offline using global information. As local information is gathered, it replans at each time instant as it is running in real-time. Grade: A.

- Computational Complexity. It has already been argued that this method can be run in real-time. However, as the complexity and size of the problem grows (as is the case in real world missions), there is no guarantee that this method can be quick enough for real-time operation. Grade: C.

- Portability. The Optimal Control method is highly portable. It can be applied to any vehicle in any environment while optimizing any parameters. It can even be adapted to follow certain paths (such as a zigzag pattern) while avoiding obstacles and ultimately reaching its target. It again comes down to proper problem formulation. Optimal Control path planning has been applied to a tricycle [1, 52], a UGV [1, 51, 52, 54], a UAV [1, 51, 52, 53, 54], an inverted pendulum problem [1, 52], the NPSAT1 Spacecraft [49, 51, 55], and an RLV [50, 51]. Grade: A.

- Completeness. A solution is guaranteed if one exists. This method can also distinguish if a solution does not exist by simply checking the feasibility of solutions as they are calculated. Grade: A.

- Multiple Vehicles. The fact that this method can handle moving obstacles automatically makes it viable for planning with multiple vehicles in a decoupled manner. Each vehicle simply treats other vehicles as moving obstacles. Going the extra step to conduct centralized planning for multiple vehicles is possible by defining a state space that simultaneously represents the configurations of all of the vehicles. Grade: A.

- Handling Errors and Uncertainties. This goes hand in hand with the ability to replan the trajectory from the current vehicle position to the target at each time instant. By using the method in real-time, the vehicle continuously updates its position and replans its trajectory, thus accounting for any positional error resulting from imperfect modeling, sensor errors and the like. Continual replanning prevents any error, deviation or uncertainty from growing too large. Refer to [1, 49-55] for successful applications of this concept. Grade: A.

D. FOCUS OF THIS WORK

Trajectory Planning, as described in [3], finds the control inputs yielding a trajectory that avoids obstacles, takes the system to the desired goal state, and may optimize some cost function while doing so. As discussed earlier in this chapter, there are two approaches to trajectory planning for a dynamic system: The decoupled approach and the direct approach. The decoupled approach involves first searching for a path (using a path planner) and then finding a time-optimal time scaling for the path subject to the actuator limits. Having studied this approach in this chapter, a few drawbacks have been uncovered. If more than one possible path exists, and in most cases there are many paths, then every path must be time-optimized to determine which path results in the best time optimal answer. If the cost function depends on state variables and controls as well as time, the optimal path may not be achievable in a decoupled manner. Centralized path planning for multiple vehicles and dynamic obstacle avoidance may not be successful if time is not considered until after path generation. Finally, conducting path planning followed by time optimization is another hybrid approach which is considered a drawback in this work as we are seeking a planner that can do it all.

The direct approach searches for the trajectory directly within the system's state space. Examples of this approach are: 1. Grid-based searches such as Dynamic Programming. [3] The drawback of this approach is that the size of the grid grows exponentially in the dimensions of the state space, making it impractical for more complex systems. 2. Randomized probabilistic methods such as RRTs [3]. This approach, already discussed in this work, trades off optimality for run-time. 3. Gradient-based numerical methods. [3] Based on APFs already discussed in this work. The difference here is that in this approach the gradient forces are applied directly to the actuators, thus a feedback law is specified for all vehicle states. The drawback here is that it shares the same disadvantages of the APF method. 4. Optimal Control with Numerical Optimization. [3]

A summary of all the path planning methods and their grades for all the parameters considered is provided as Table 1. It is clear from this compiled data that the best approach for path planning, when considering the desired performance parameters, is the Optimal Control approach. The most telling fact is that the Optimal Control method does not need help from other techniques to achieve the desired performance. This is important in that hybrid techniques will often suffer the shortcomings of both methods. The focus of this research will be in studying the utility of the Optimal Control technique in the real-time autonomous trajectory planning of Unmanned Ground Vehicles (UGVs). Specifically, this work will address the following issues:

- The determination of grades for the various performance parameters did not take into account the fact that sometimes in order to achieve satisfactory performance in one area, a reduction in performance (or even failure) in another area would be the cost. Table 1 shows that Optimal Control received the best grades, but can the Optimal Control technique conduct path planning while simultaneously achieving grade 'A' performances in all the desired performance parameters? The answer to this question will become apparent as scenarios are presented and studied throughout this work.

Table 1. Summary of Path Planners with Grades.

	Path Planning and Control	Optimality	Obstacle Avoidance	Handling Vehicle Constraints	Global vs Local Info	Computational Complexity	Portability	Completeness	Multiple Vehicles	Errors and Uncertainties
Bug Algorithms	**C**	**F**	**F**	**C**	**C**	**A**	**C**	**A**	**F**	**C**
Potential Fields	**C**	**C**	**A**	**C**	**A**	**C**	**C**	**A**	**A**	**A**
Roadmaps	**C**	**F**	**A**	**A**	**A**	**C**	**C**	**A**	**A**	**A**
Cell Decomposition	**C**	**C**	**C**	**C**	**C**	**F**	**C**	**A**	**C**	**C**
Optimal Control	**A**	**A**	**A**	**A**	**A**	**C**	**A**	**A**	**A**	**A**

- The speed of the Optimal Control method will be considered throughout all the scenarios studied in this work, but only for the purpose of comparison. Computational complexity was the only parameter given a ‘C’ grade in the optimal control technique. This is because even if sufficient speed can be shown for various scenarios, there will always be a higher and higher demand for more complex missions that will require faster speeds. This work does not prove real-time operation is possible in all cases; but it shows how powerful the optimal control technique can be, assuming real-time operation. The speed of the algorithm is paramount to being able to accomplish real-time operation, which implies grade 'A' performance in dynamic obstacle avoidance (no prediction used), computational complexity and the ability to handle errors and uncertainties. The definition of real-time varies depending on the application. For example, a jet aircraft would require a planner that can calculate a trajectory in a matter of milliseconds. For the purposes of this work with UGVs, real-time operation signifies the ability to replan a trajectory in less than *0.5 seconds*.
- The ability to apply Optimal Control path planning in the presence of dynamic obstacles will be studied. It has already been shown that this method is viable when the future positions of moving obstacles are known, such as when the complete trajectories of obstacles are known [1, 52]. But, what of its ability to plan around obstacles whose movements cannot be completely predicted? This aspect lends itself to the ability to conduct non-cooperative motion planning. The planner must be able to replan using the latest snapshot of the environment at each time instant. This goes hand-in-hand with the ability to operate in real-time.
- The same line of thinking for moving obstacles will also be applied to a moving target. Can the planner achieve success when the target changes in mid-mission? Can the vehicle rendezvous with the target when the target is in motion? What if that motion is not known *a priori*?
- The portability of the Optimal Control technique will be studied further. Can the vehicle follow a specific path (that may not be optimal) without the addition of a trajectory following subsystem? Can the planner calculate a trajectory based on safety factors or the preferred routes of the user?

- The ability to plan trajectories for multiple vehicles will be studied (cooperative motion planning). If dynamic obstacle avoidance is possible, then treating other vehicles as moving obstacles is possible as well. However, this work goes the extra step by investigating centralized planning for multiple vehicles. This is done by defining a state space that simultaneously represents the configurations of all of the vehicles. The issue then becomes how to manage the architecture. Does each vehicle have its own path planner and does it receive communications of course and speeds from other vehicles? Or does one vehicle's onboard computer conduct path planning and communicate the trajectories to the other vehicles? Should one vehicle be considered the lead vehicle and not have to account for the other vehicles? What level of cooperation is needed between vehicles?

III. OPTIMAL CONTROL PROBLEM FORMULATION

A. BASIC FORM OF THE OPTIMAL CONTROL PROBLEM

Fundamental to any Optimal Control problem is proper problem formulation. This includes deriving a set of functions and equations that represent the kinodynamic equations of the system, the path constraints and the objective (cost) function. These equations must be sufficiently smooth. In other words, they must be continuous and differentiable. The kinodynamic equations are converted into state space form and define the relationship between the system state and the control input. The path constraints indicate constraints on the state and control variables in addition to their maximum and minimum limits; for example, they can represent equations used to model obstacles for obstacle avoidance in trajectory planning. The obstacles need not be smooth, but the constraint functions used to define the obstacles must be made up of one or more smooth functions. The cost function can be any function of state variables, control variables and time. The other elements that make up the problem formulation are the endpoint constraints (also known as the state variable initial and final conditions). The remainder of Chapter III.A covers the basic Optimal Control problem formulation/analysis and was taken from [48, 56, 63]. Table 2 provides a breakdown of the notation.

Table 2. Optimal Control Problem Notation.

Symbol	**Description**
$\underline{x}$	N_x dimensional state vector
$\underline{u}$	N_u dimensional control vector
$\underline{x}(\cdot), \underline{u}(\cdot)$	system state and control trajectories
$\underline{x}*(\cdot), \underline{u}*(\cdot)$	optimal system trajectory
t_0	initial time
t_f	final time
$\underline{x}_0, \underline{x}_f$	state at the initial and final times
J	objective (cost) function
E, F	endpoint and running costs
$\underline{f}$	state equations
$\underline{h}, \underline{e}$	path and endpoint constraints
$\underline{h}^L, \underline{e}^L$	lower bounds on path and endpoint constraints
$\underline{h}^U, \underline{e}^U$	upper bounds on path and endpoint constraints
H	Hamiltonian
$\underline{\lambda}$	Lagrange multiplier associated with $\underline{f}$ (costate)
$\bar{H}$	Lagrangian of H
$\underline{\mu}$	Lagrange multiplier associated with $\underline{h}$
$\bar{E}$	endpoint Lagrangian
$\underline{\nu}$	Lagrange multiplier associated with $\underline{e}$

1. The Fundamental Optimal Control Problem Statement

The Optimal Control trajectory planning problem is to find the control solution to the state equations that takes the vehicle from its initial state at time zero to the target state at some final time, while avoiding obstacles, obeying vehicle state and control limits, and minimizing some cost function. It is provided here in its most general form as equation set (1).

$$
\begin{array}{ll}
\textit{Given} & \begin{array}{l} \underline{x} \in X \subset \mathbb{R}^{N_x} \\ \underline{u} \in U \subset \mathbb{R}^{N_u} \end{array} \\[2ex]
\textit{Find} & \underline{x}(\cdot), \underline{u}(\cdot) \\[2ex]
\textit{That Minimizes} & J\left[\underline{x}(\cdot), \underline{u}(\cdot), t_0, t_f\right] = E\left(\underline{x}_0, \underline{x}_f, t_0, t_f\right) + \int_{t_0}^{t_f} F\left(\underline{x}(t), \underline{u}(t), t\right) dt \quad (1) \\[2ex]
\textit{Subject to} & \begin{array}{l} \underline{\dot{x}}(t) = \underline{f}\left(\underline{x}(t), \underline{u}(t), t\right) \\ \underline{h}^L \le \underline{h}\left(\underline{x}(t), \underline{u}(t), t\right) \le \underline{h}^U \\ \underline{e}^L \le \underline{e}\left(\underline{x}_0, \underline{x}_f, t_0, t_f\right) \le \underline{e}^U \end{array}
\end{array}
$$

2. Scaling and Balancing

It is important in the problem formulation to scale the variables. The choice of scaling will balance the equations for numerical analysis, thus improving the accuracy of the answer and the computation time. How a problem can be scaled and balanced will be demonstrated later when the Optimal Control technique is applied to a 4-wheel car.

3. Verification and Validation

The Minimum Principle converts the infinite dimensional optimal control problem to an instantaneous finite dimensional mathematical programming problem in the control parameter $\underline{u}$. [63] This can be numerically solved using DIDO [56] to generate an extremal. Validating a solution is in fact an extremal is difficult, but can be done by first showing the feasibility of the generated solution, then verifying Pontryagin's Principle, and finally verifying Bellman's Principle.

Showing the feasibility of the generated solution can be done by control trajectory interpolation and state propagation using a Runge-Kutta algorithm. If the initial conditions and system dynamics can be propagated using the optimal control solution and it matches the DIDO generated trajectories, then the control solution is deemed feasible.

According to Pontryagin's Minimum Principle, the following five necessary (but not sufficient) conditions for optimality must be met: The Adjoint Equation, the Hamiltonian Minimization Condition, the Transversality Condition, the Hamiltonian Value Condition and the Hamiltonian Evolution Equation. In order to verify these conditions, we must first define the Hamiltonian (*H*):

$$H\left(\underline{\lambda},\underline{x},\underline{u},t\right)=F\left(\underline{x},\underline{u},t\right)+\underline{\lambda}^{T}\,\underline{f}\left(\underline{x},\underline{u},t\right) \tag{2}$$

The Hamiltonian Minimization Condition (HMC) minimizes *H* with respect to $\underline{u}$ while holding $\underline{\lambda}$ and $\underline{x}$ constant. The HMC formulation appears in equation set (3).

$$\begin{aligned} &\textit{Minimize}_{\underline{u}} && H\left(\underline{\lambda},\underline{x},\underline{u},t\right) \\ &\textit{Subject to} && \underline{u}\in U \\ & && \underline{h}^{L}\le \underline{h}\left(\underline{x}(t),\underline{u}(t),t\right)\le \underline{h}^{U} \end{aligned} \tag{3}$$

We must take into account the path constraints that are a function of $\underline{u}$. We do this by finding the Lagrangian of the Hamiltonian ($\overline{H}$):

$$\overline{H}\left(\underline{\mu},\underline{\lambda},\underline{x},\underline{u},t\right)=H\left(\underline{\lambda},\underline{x},\underline{u},t\right)+\underline{\mu}^{T}\,\underline{h}\left(\underline{x},\underline{u},t\right) \tag{4}$$

The HMC then becomes:

$$\frac{\partial\overline{H}}{\partial\underline{u}}=\underline{0} \tag{5}$$

The Karush-Kuhn-Tucker (KKT) conditions for the covectors, $\underline{\mu}$, follow as:

$$\underline{\mu}\begin{cases}\le 0 & \\ =0 & \textit{for} \\ \ge 0 & \end{cases}\begin{matrix} h\left(\underline{x},\underline{u},t\right)=h^{L} \\ h^{L}<h\left(\underline{x},\underline{u},t\right)<h^{U} \\ h\left(\underline{x},\underline{u},t\right)=h^{U}\end{matrix} \tag{6}$$

The costate ($\underline{\lambda}$) must satisfy equation (7), the Adjoint Equation.

$$-\dot{\underline{\lambda}} = \frac{\partial \overline{H}}{\partial \underline{x}} \tag{7}$$

The Endpoint Lagrangian ($\overline{E}$) is defined in equation (8).

$$\overline{E}\left(\underline{\nu}, \underline{x}_0, \underline{x}_f, t_0, t_f\right) = E\left(\underline{x}_0, \underline{x}_f, t_0, t_f\right) + \underline{\nu}^T \underline{e}\left(\underline{x}_0, \underline{x}_f, t_0, t_f\right) \tag{8}$$

This is used to establish the Transversality Condition of equation set (9).

$$\begin{aligned} \underline{\lambda}(t_f) &= \frac{\partial \overline{E}}{\partial \underline{x}(t_f)} \\ \underline{\lambda}(t_0) &= \frac{-\partial \overline{E}}{\partial \underline{x}(t_0)} \end{aligned} \tag{9}$$

Another Transversality Condition called the Hamiltonian Value Condition provides the necessary condition for when the final or initial times are not fixed. It is shown here as equation set (10).

$$\begin{aligned} H(t_f) &= \frac{-\partial \overline{E}}{\partial t_f} \\ H(t_0) &= \frac{\partial \overline{E}}{\partial t_0} \end{aligned} \tag{10}$$

The value of the Hamiltonian over the optimal solution ($\underline{u}^*$) will be a minimum and is called the Lower Hamiltonian ($\mathrm{H} = H(\underline{u}*)$). The Hamiltonian Evolution Equation is given as equation (11).

$$\dot{\mathrm{H}} = \frac{\partial \overline{H}}{\partial t} \tag{11}$$

Bellman's Principle of optimality simply states that given an optimal trajectory from point A to point B, and a point C lying on that trajectory, the cost to go from A to C plus the cost from C to B will equal the cost from A to B. Likewise, the independently generated trajectories from A to C plus C to B will be identical to the trajectory from A to B. Bellman's Principle can be verified by picking one or more points along the optimal trajectory and showing that the pieces add up to the whole, both in the cost of the maneuver and in the actual trajectory.

B. THE FOUR-WHEELED CAR

The four-wheeled car system (UGV) discussed in this work has rear-wheel drive and front-wheel steering. It uses the Ackerman steering configuration common to the standard automobile. This configuration assumes all the wheels turn around the same point (rotation center), which is colinear with the rear axle of the car. Only the front wheels are capable of turning and the back wheels must roll without slipping[2]. This vehicle model choice was made in part because the standard car is the most relied upon means of transportation in our society. It is the first nonholonomic system ever studied in robotics [64]. In the world of autonomous vehicles, it is not so prevalent, because there are many designs that provide greater mobility than the standard four-wheeled car. The motion of a four-wheeled car with front-wheel steering is too constrained for real world autonomous applications. It is common sense to say that the best vehicle for such applications would have the highest degree of mobility, like being able to turn in place or move sideways. The model in this work requires a parking-type maneuver to move sideways; i.e., a repeated change in direction while switching between forward and backward motion.

The nonholonomic constrained motion inherent in the four-wheeled car system is the biggest reason why it is the system of choice for this work. If autonomous trajectory planning can be achieved on this system, then it can be achieved on any less constrained system. Some other, simpler, versions of the four-wheel car have been used for experimentation in path planning. The Reeds-Shepp car [65] severely limits the velocity choices by bounding velocity to $|v| = 1$. It is capable of only maximum forward or maximum reverse speeds. The Dubins vehicle [66] limits velocity even further to v=1, thus only allowing maximum forward speed.

The standard kinematic car model [1, 2, 3] that is used in this work is shown in the remainder of Chapter III.B. Figure 20 shows the model configuration [1]. L is the length of the car between the front and rear axles. r_t is the instantaneous turning radius.

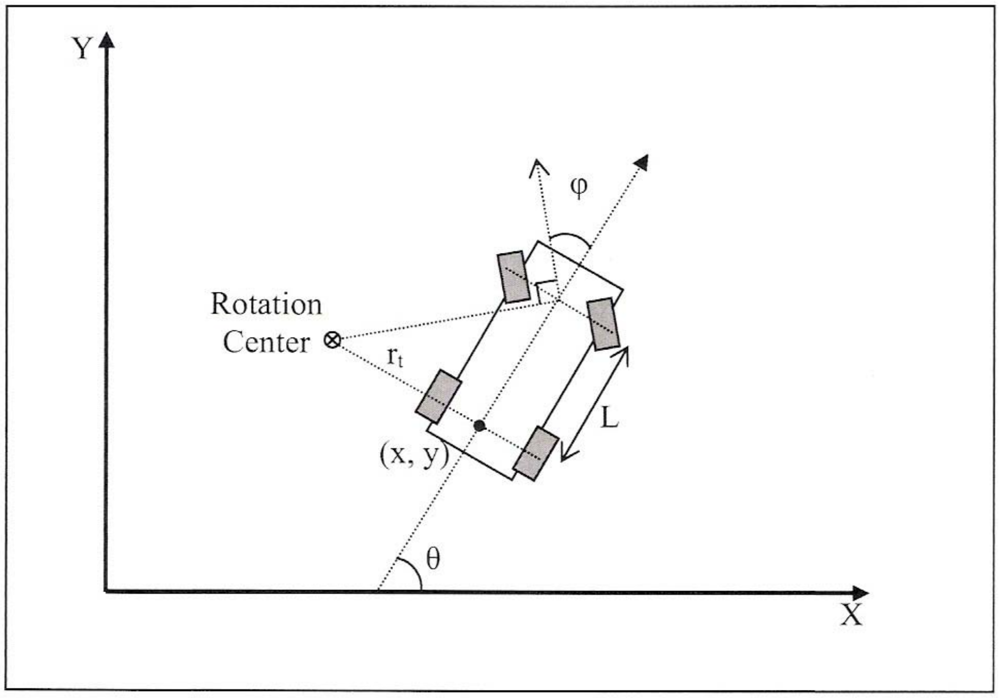


Figure 20. Four-wheeled Car Model with Front-wheel Steering. [From 1]

The state vector is composed of two position variables (*x*,*y*) and an orientation variable (θ). The *x*-*y* position of the car is measured at the center point of the rear axle.

$$\underline{x} \in X := \begin{Bmatrix} x : x_{\min} \le x(t) \le x_{\max} \\ y : y_{\min} \le y(t) \le y_{\max} \\ \theta : \theta_{\min} \le \theta(t) \le \theta_{\max} \end{Bmatrix}$$

The control vector consists of the vehicle's velocity (*v*) and the angle of the front wheels (φ) with respect to the car's heading:

$$\underline{u} \in U := \begin{Bmatrix} v : v_{\min} \le v(t) \le v_{\max} \\ \varphi : \varphi_{\min} \le \varphi(t) \le \varphi_{\max} \end{Bmatrix}$$

Using the fact that $L = r_t \tan(\varphi)$ and $v = r_t\dot{\theta}$ results in the kinematic equations of motion for the UGV:

$$\underline{\dot{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \dfrac{v}{L}\tan(\varphi) \end{bmatrix}$$

It is very important to note here that the problem with using v,φ as the control vector is that the control solution put forth by DIDO is not smooth, is subject to instantaneous jumps in value, and therefore is very difficult to interpolate and use as the controls for state propagation purposes. It is also true that in an actual vehicle the speed and wheel angle cannot change instantaneously. For these reasons, in order to smooth out the controls (v,φ) and to be able to apply the results of this work to an actual vehicle in the future, an added level of control was included. In other words, v,φ was included as part of the state vector, and a,ω was added as the new control vector. This creates smoother trajectories for v,φ that can be used in an actual vehicle. For purposes of this work, DIDO will use the state and control vectors of equations (12) and (13). However, v,φ will be used as the control vector for a vehicle with an x,y,θ state vector when conducting the feasibility checks. The added layer of control transforms the state vector to:

$$\underline{x} \in X := \begin{Bmatrix} x : x_{\min} \le x(t) \le x_{\max} \\ y : y_{\min} \le y(t) \le y_{\max} \\ \theta : \theta_{\min} \le \theta(t) \le \theta_{\max} \\ v : v_{\min} \le v(t) \le v_{\max} \\ \varphi : \varphi_{\min} \le \varphi(t) \le \varphi_{\max} \end{Bmatrix} \tag{12}$$

The new control vector consists of the vehicle's acceleration (a) and the rate of change of the front wheel angle (ω) (equation (13)).

$$\underline{u} \in U := \begin{Bmatrix} a : a_{\min} \le a(t) \le a_{\max} \\ \omega : \omega_{\min} \le \omega(t) \le \omega_{\max} \end{Bmatrix} \tag{13}$$

The updated equations of motion for the UGV become:

$$\underline{\dot{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \\ \dot{\varphi} \end{bmatrix} = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \frac{v}{L}\tan(\varphi) \\ a \\ \omega \end{bmatrix} \tag{14}$$

C. OPTIMAL CONTROL PROBLEM FOR THE FOUR-WHEELED CAR

In order to formulate the optimal control problem we must first establish the vehicle parameters, the state and control limits, the constraints, the cost function, and the desired task. The vehicle parameter that is needed is the length (L) between the front and rear axles of the vehicle. Unless stated otherwise, the distance between the axles will be half a meter, or $L = 0.5m$. The state and control limits are defined as follows:

$$\underline{x} \in X := \begin{Bmatrix} x : 0 \le x(t) \le 10 \\ y : 0 \le y(t) \le 10 \\ \theta : -3\pi \le \theta(t) \le 3\pi \\ v : -1 \le v(t) \le 1 \\ \varphi : -1 \le \varphi(t) \le 1 \end{Bmatrix} \tag{15}$$

$$\underline{u} \in U := \begin{Bmatrix} a : -0.5 \le a(t) \le 0.5 \\ \omega : -0.33 \le \omega(t) \le 0.33 \end{Bmatrix} \tag{16}$$

The units of distance will be in *m*, time in *s*, and angle in *rads*. The vehicle's task will be to move sideways to the right *1m* (from (x_0, y_0) =(5,5) to (x_f, y_f) =(5,4)) in minimum time. Its initial and final states will be stopped ($v = 0$), facing to the right ($\theta = 0$) with zero wheel angle ($\varphi = 0$). Figure 21 shows the initial and final vehicle configurations. No other constraints will be specified for this example problem.

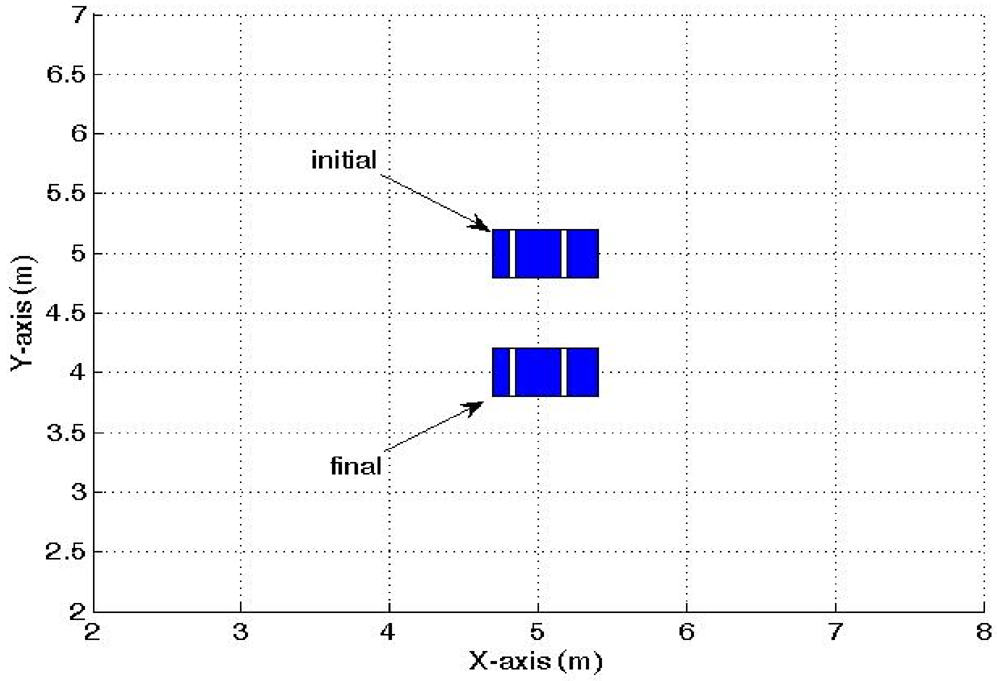


Figure 21. Configurations for Initial Vehicle Problem.

1. Optimal Control Problem Statement

The state ($\underline{x}$) and control ($\underline{u}$) vectors are given by equations (15) and (16) respectively. The state equations are given by equation (14) with $L = 0.5m$. The cost function is established by the fact that the given maneuver must be completed in minimum time. No other requirements are made. This results in an endpoint cost ($E\left(\underline{x}_0, \underline{x}_f, t_0, t_f\right) = t_f$) and no running cost ($F\left(\underline{x}(t), \underline{u}(t), t\right) = 0$). The task now is to formulate the path and endpoint constraints. The limits on *x* can be converted into a path constraint by assigning $h_i = x(t)$, resulting in $h_i^L = x_{\min}$ and $h_i^U = x_{\max}$. The initial and final conditions on *x* can be converted into endpoint constraints by assigning $e_i^L = e_i^U = e_i = x(t_0) - x_0 = 0$ and $e_j^L = e_j^U = e_j = x(t_f) - x_f = 0$. The remaining state and control variables can be similarly encoded into path and endpoint constraints. The full optimal control problem statement becomes equation set (17).

Given $\underline{x} = \begin{bmatrix} x \\ y \\ \theta \\ v \\ \varphi \end{bmatrix} \in X \subset \mathbb{R}^5$ $\underline{u} = \begin{bmatrix} a \\ \omega \end{bmatrix} \in U \subset \mathbb{R}^2$

Find $\underline{x}(\cdot), \underline{u}(\cdot)$

That Minimizes $J = t_f$

Subject to
$$\dot{\underline{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \\ \dot{\varphi} \end{bmatrix} = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ 2v\tan(\varphi) \\ a \\ \omega \end{bmatrix} \tag{17}$$

$$\begin{gathered} 0 \le x(t) \le 10 \\ 0 \le y(t) \le 10 \\ -3\pi \le \theta(t) \le 3\pi \\ -1 \le v(t) \le 1 \\ -1 \le \varphi(t) \le 1 \end{gathered} \qquad \begin{gathered} -0.5 \le a(t) \le 0.5 \\ -0.33 \le \omega(t) \le 0.33 \end{gathered}$$

$$e(\underline{x}_0, \underline{x}_f, t_0, t_f) = \begin{bmatrix} t_0 \\ x(t_0) - 5 \\ y(t_0) - 5 \\ \theta(t_0) \\ v(t_0) \\ \varphi(t_0) \\ x(t_f) - 5 \\ y(t_f) - 4 \\ \theta(t_f) \\ v(t_f) \\ \varphi(t_f) \end{bmatrix} = [\underline{0}]$$

2. Scaling and Balancing

It has already been stated that the choice of scaling will balance the equations for numerical analysis, thus improving the accuracy of the answer and the computation time. The state and control limits used in the current problem formulation already provide a well balanced set of equations, but what if one of the variables was a couple of orders of magnitude greater than the others? For example, what if the range on *x* and *y* was $0 \le x(t) \le 1000$ and $0 \le y(t) \le 1000$ respectively? By using a scaling factor (D), we can scale down *x* and *y* as shown in equation set (18).

$$\begin{aligned} \bar{x} &= \frac{x}{D} \\ \bar{y} &= \frac{y}{D} \end{aligned} \tag{18}$$

The limits on the new scaled variables, and therefore the new path constraints will be:

$$\begin{aligned} 0 \le \bar{x}(t) \le 1000/D \\ 0 \le \bar{y}(t) \le 1000/D \end{aligned} \tag{19}$$

The changes to the state equations can be written as:

$$\underline{\dot{x}} = \begin{bmatrix} \dot{\bar{x}} \\ \dot{\bar{y}} \end{bmatrix} = \begin{bmatrix} \frac{v}{D}\cos(\theta) \\ \frac{v}{D}\sin(\theta) \end{bmatrix} \tag{20}$$

Similarly, the changes to the endpoint constraints become:

$$e(\underline{\bar{x}}_0, \underline{\bar{x}}_f, t_0, t_f) = \begin{bmatrix} \bar{x}(t_0) - \frac{x_0}{D} \\ \bar{y}(t_0) - \frac{y_0}{D} \\ \bar{x}(t_f) - \frac{x_f}{D} \\ \bar{y}(t_f) - \frac{y_f}{D} \end{bmatrix} = [\underline{0}] \tag{21}$$

The above formulation for scaling variables is not needed for the current example, but is shown for completeness and to show how scaling might be performed later in this work if it is deemed necessary for future scenarios.

3. Verification and Validation

The optimal control problem of Figure 21 was solved numerically using DIDO [56]. A 100 node solution was obtained, and the overall maneuver time (which is to say the overall cost, since it is a minimum time problem) was 8.07 *sec*. Figure 22 shows the complete maneuver, while Figures 23 and 24 show the state and control trajectories respectively. The purpose here is to show (using feasibility, Pontryagin's Minimum Principle, and Bellman's Principle) that the solution is an extremal. Notice the non-smoothness of the control variables in Figure 24. This is why an added layer of control was included, resulting in a,ω being the control vector vice v,φ. The smooth results obtained for v,φ as part of the state vector can now be interpolated and used as the controls of an actual vehicle with an x,y,θ state.

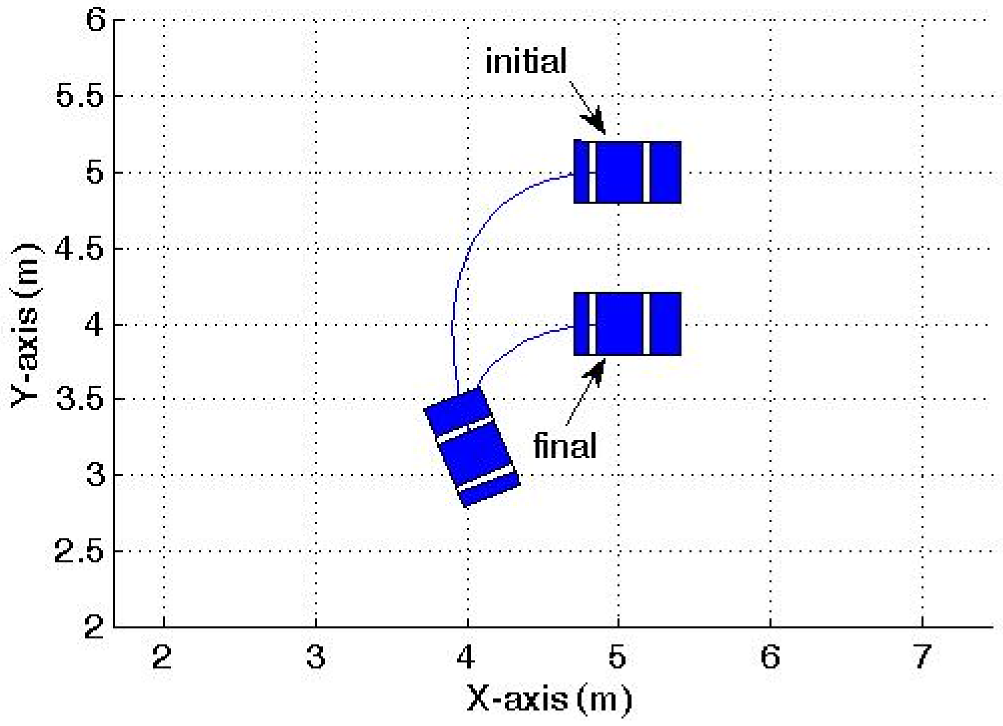


Figure 22. Sideways Maneuver, Same Orientation.

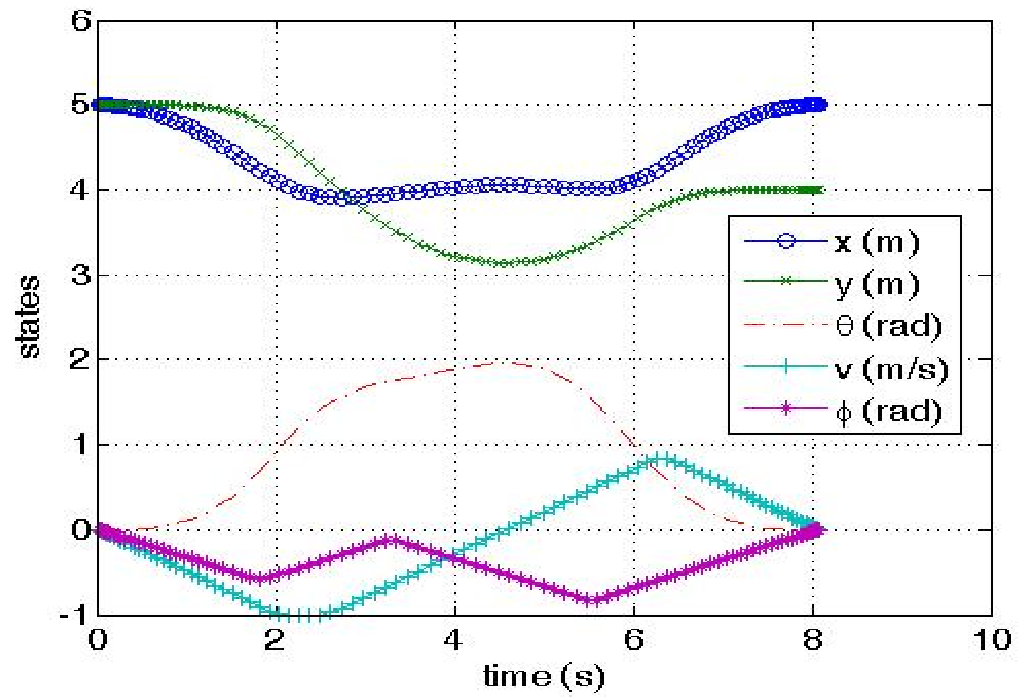


Figure 23. State Trajectory.

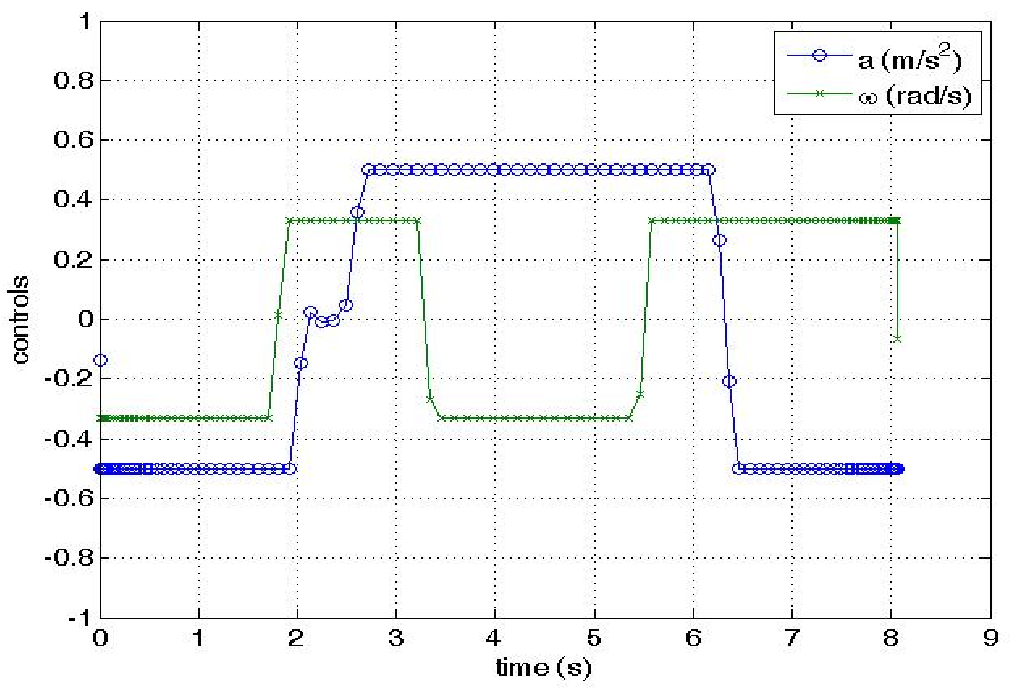


Figure 24. Control Trajectory.

The first step in verification is that of showing the feasibility of the generated solution. The v, φ state variables were used as the control solution. The initial conditions and system dynamics of the x, y, θ state variables were propagated using the v, φ control solution. Figures 25 and 26 show the propagated x vs y and θ vs t trajectories superimposed on the DIDO solutions, and they are nearly identical, which tells us that the optimal solution is feasible.

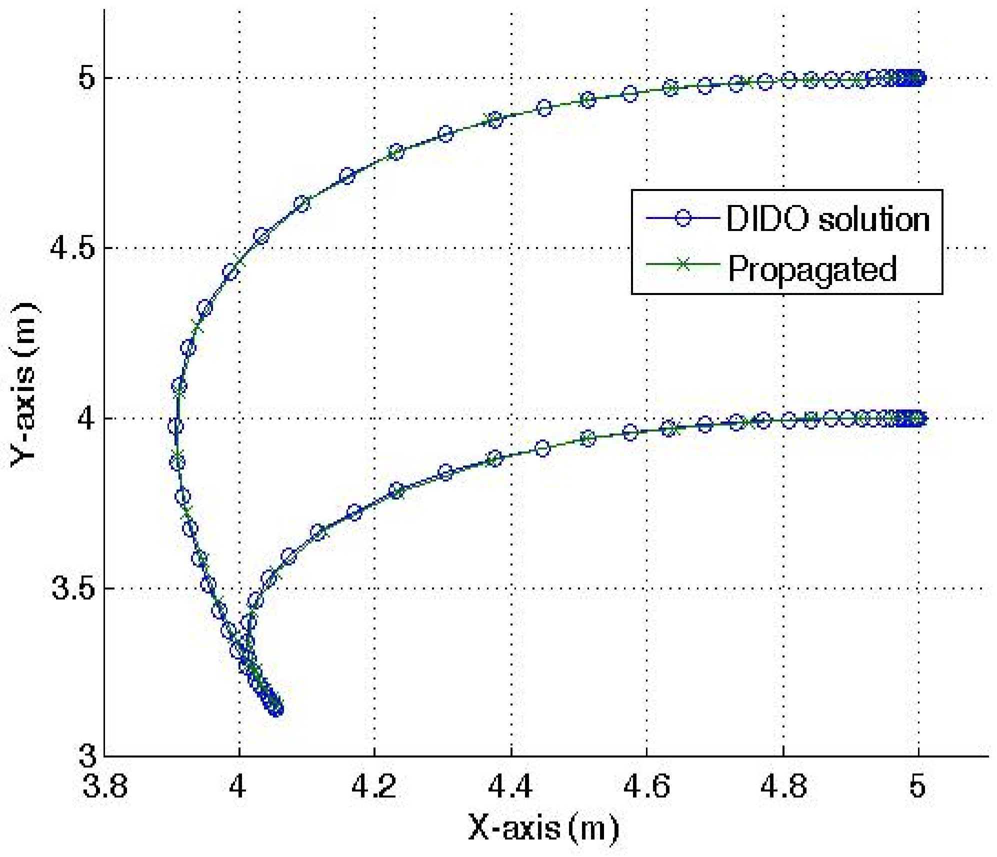


Figure 25. Feasibility Check (*x* vs *y*).

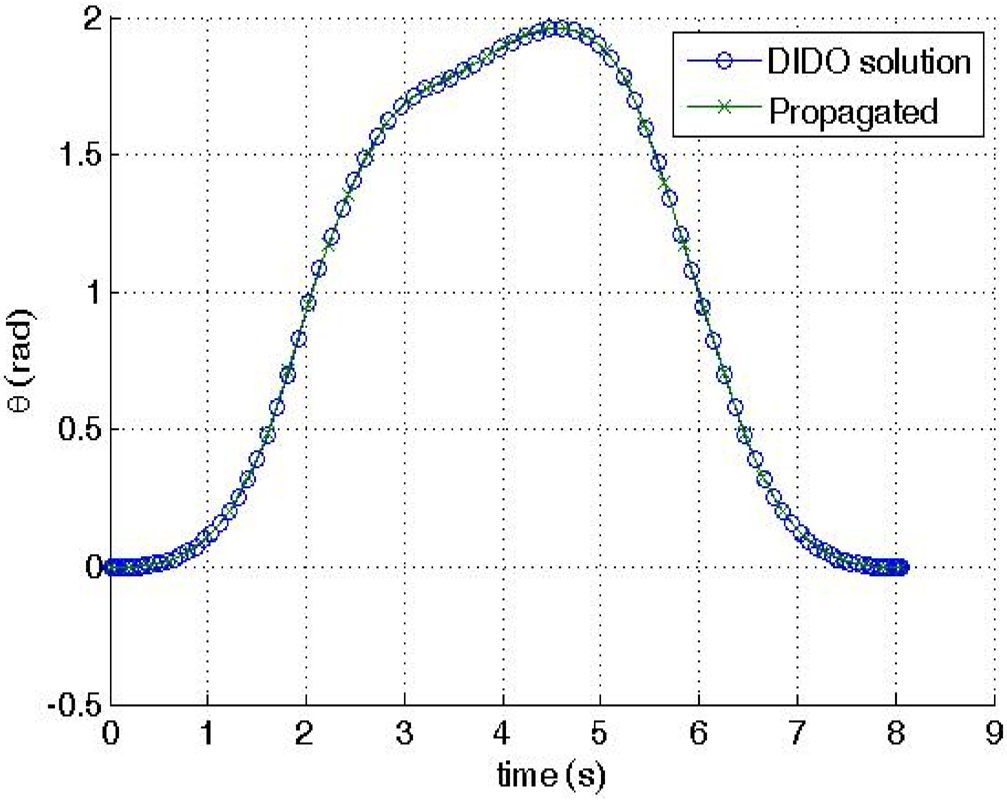


Figure 26. Feasibility Check (θ vs t).

Verifying Pontryagin's Minimum Principle involves first obtaining the Hamiltonian (H):

$$H = \lambda_x v\cos(\theta) + \lambda_y v\sin(\theta) + \lambda_\theta 2v\tan(\varphi) + \lambda_v a + \lambda_\varphi \omega \qquad (22)$$

The Lagrangian of the Hamiltonian ($\bar{H}$) becomes:

$$\bar{H} = H + \mu_x x + \mu_y y + \mu_\theta \theta + \mu_v v + \mu_\varphi \varphi + \mu_a a + \mu_\omega \omega \qquad (23)$$

The KKT conditions, equation set (6), require the state and control covectors ($\underline{\mu}$) to be less than or equal to zero when their corresponding variable is at its minimum, greater than or equal to zero when at its maximum, and equal to zero when their corresponding variable is inside its minimum/maximum range. From Figure 23 it can be seen that all the state variables (except v) stay within their limits. Figure 27 shows that all the state covectors (except μ_v) are zero, as expected. The KKT conditions require the state covector μ_v be less than or equal to zero during the time frame that *v* is at its minimum limit; μ_v and *v* have been plotted together on Figure 28 to show this relationship. It follows when comparing Figure 29 (the control covectors) to Figure 24 that the covectors oscillate between positive and negative as the control variables switch between maximum and minimum values, as again required by the KKT conditions. Figure 30 combines Figures 24 and 29 to better present these relationships.

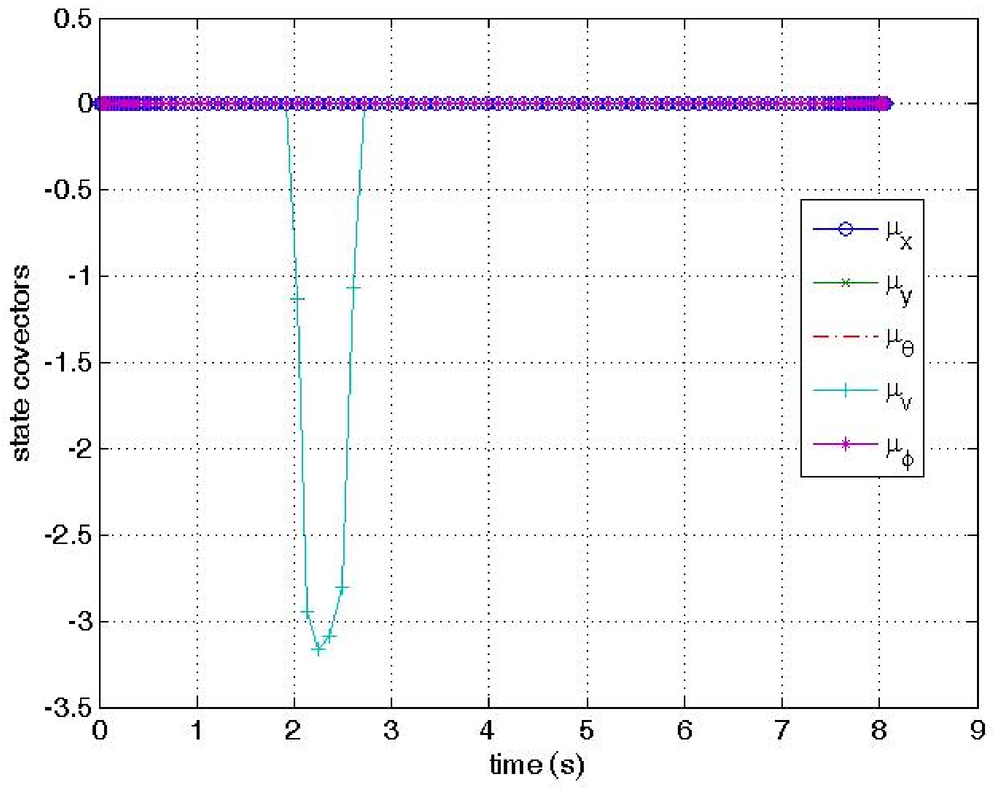


Figure 27. State Covectors.

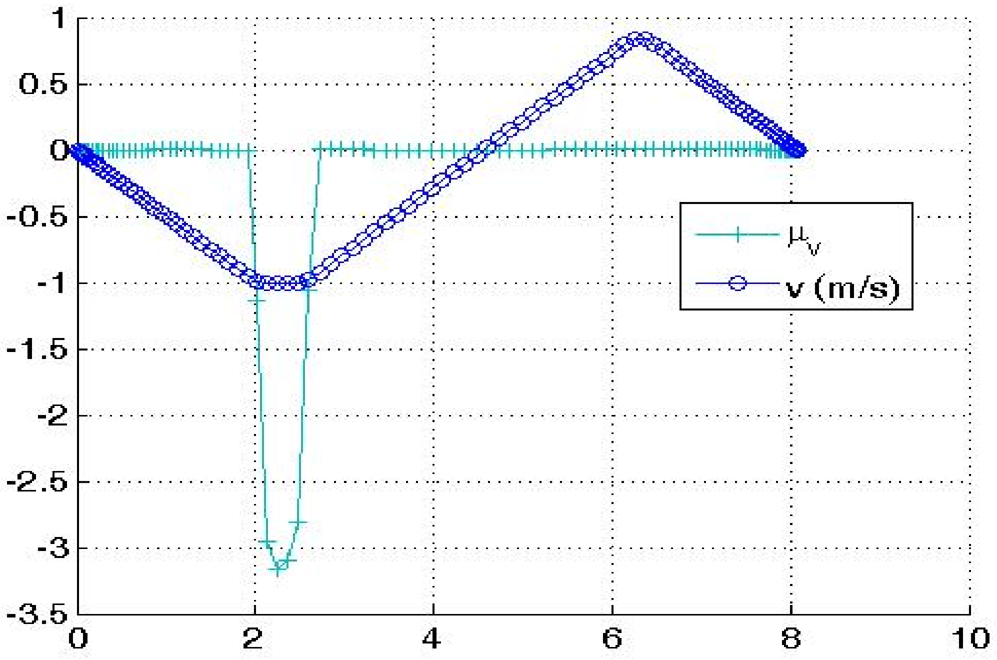


Figure 28. Relationship between μ_v and *v*.

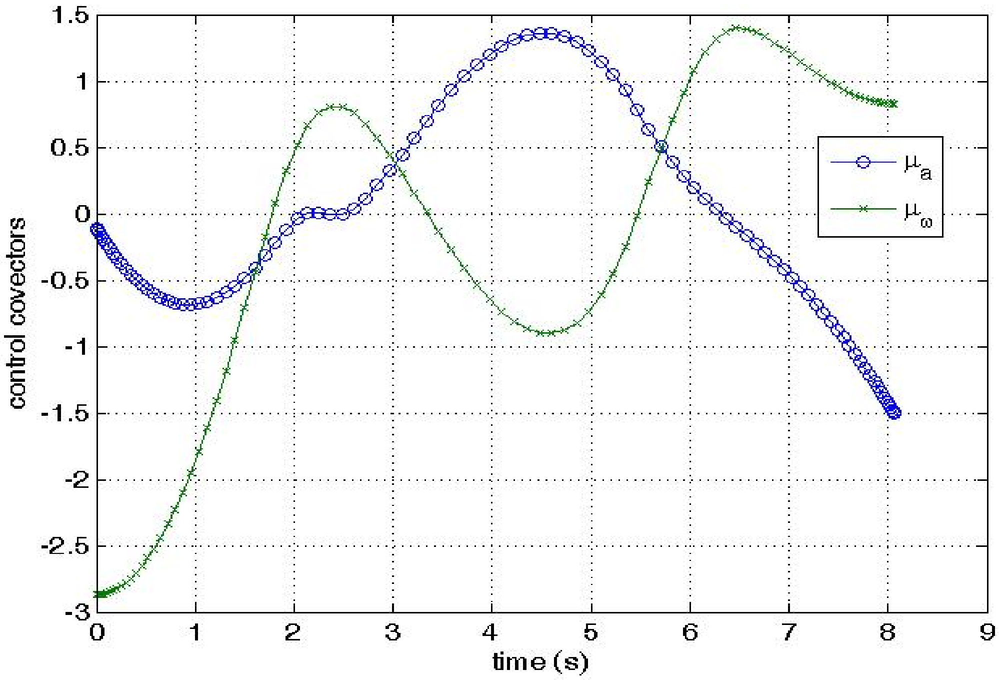


Figure 29. Control Covectors.

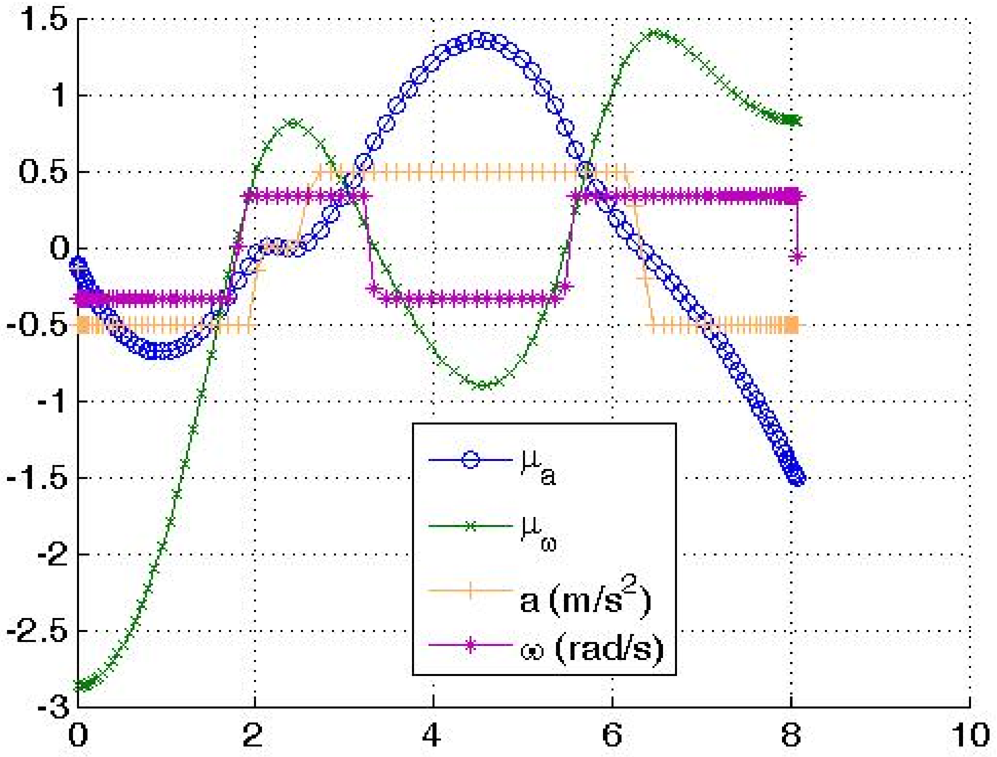


Figure 30. Relationship between μ_a, μ_ω and a, ω.

Applying the HMC results in the following:

$$\begin{aligned} \lambda_v &= -\mu_a \\ \lambda_\varphi &= -\mu_\omega \end{aligned} \tag{24}$$

Figure 31 shows the costates for this problem. Comparing the costates $\lambda_v, \lambda_\varphi$ to the control covectors of Figure 29 validates equation set (24). Figure 32 combines the graphs of $\lambda_v, \lambda_\varphi$ with those of μ_a, μ_ω to show this relation.

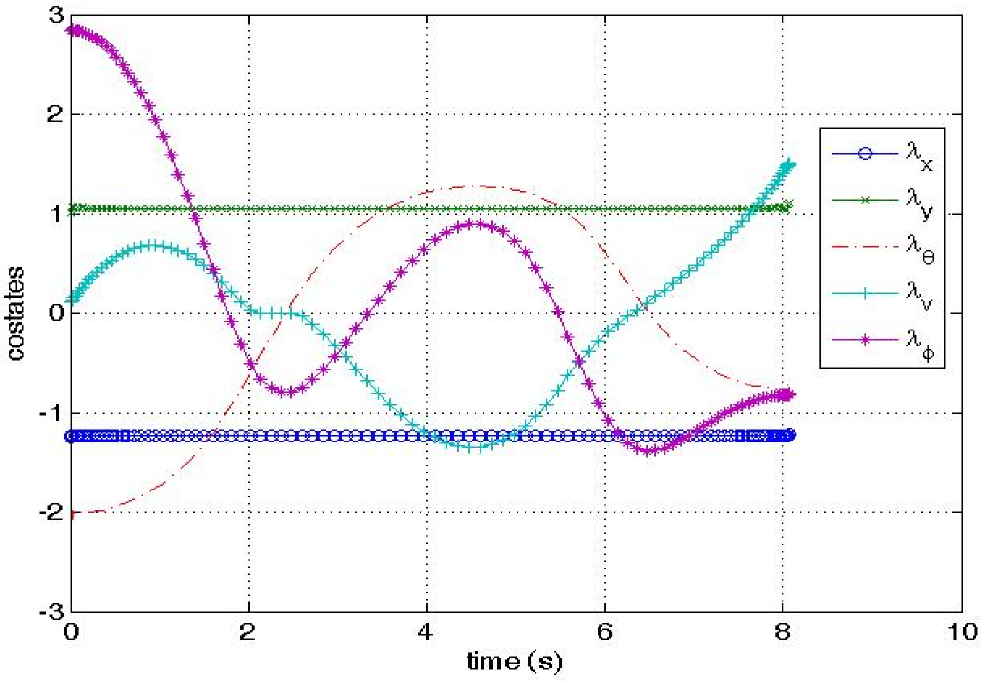


Figure 31. Costates.

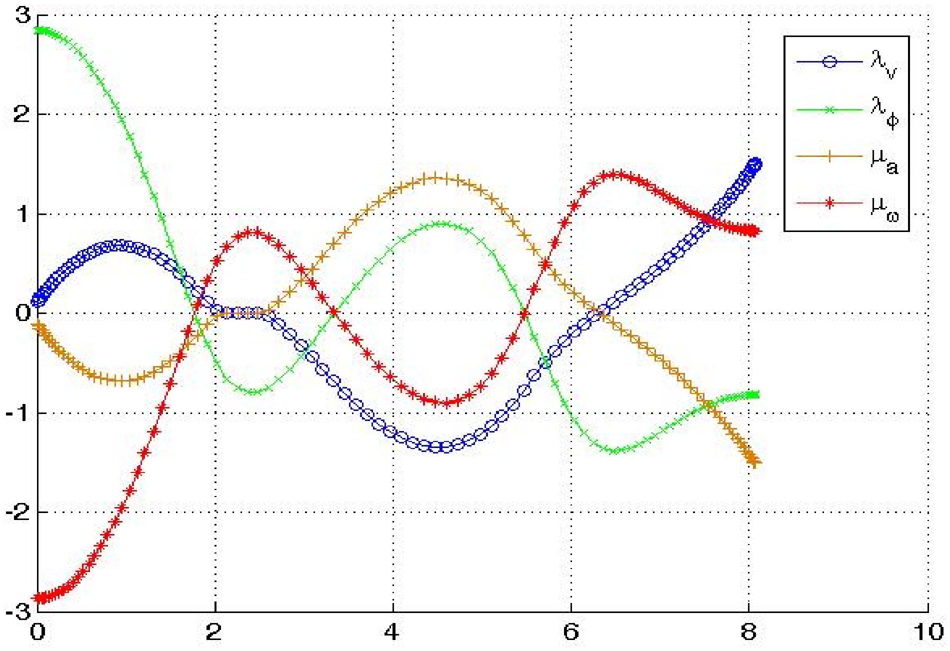


Figure 32. Comparison of $\lambda_v, \lambda_\varphi$ with μ_a, μ_ω.

Calculating the Adjoint Equation results in:

$$\begin{aligned}
\dot{\lambda}_x &= -\mu_x \\
\dot{\lambda}_y &= -\mu_y \\
\dot{\lambda}_\theta &= \lambda_x v \sin(\theta) - \lambda_y v \cos(\theta) - \mu_\theta \\
\dot{\lambda}_v &= -\lambda_x \cos(\theta) - \lambda_y \sin(\theta) - 2\lambda_\theta \tan(\varphi) - \mu_v \\
\dot{\lambda}_\varphi &= -2v\lambda_\theta \sec^2(\varphi) - \mu_\varphi
\end{aligned} \tag{25}$$

Since μ_x, μ_y are always zero (from Figure 27), it follows from equation set (25) that λ_x, λ_y are constants, and this can be validated using Figure 31. It can also be interpreted from equation set (25) that λ_θ will be constant whenever *v=0* since $\mu_\theta = 0$ (from Figure 27). This can be shown by comparing Figures 23 and 31, and noting that the graph for λ_θ approaches a local maximum or minimum whenever the velocity of the vehicle is at or crosses the zero axis. This relationship between λ_θ and *v* is shown in Figure 33.

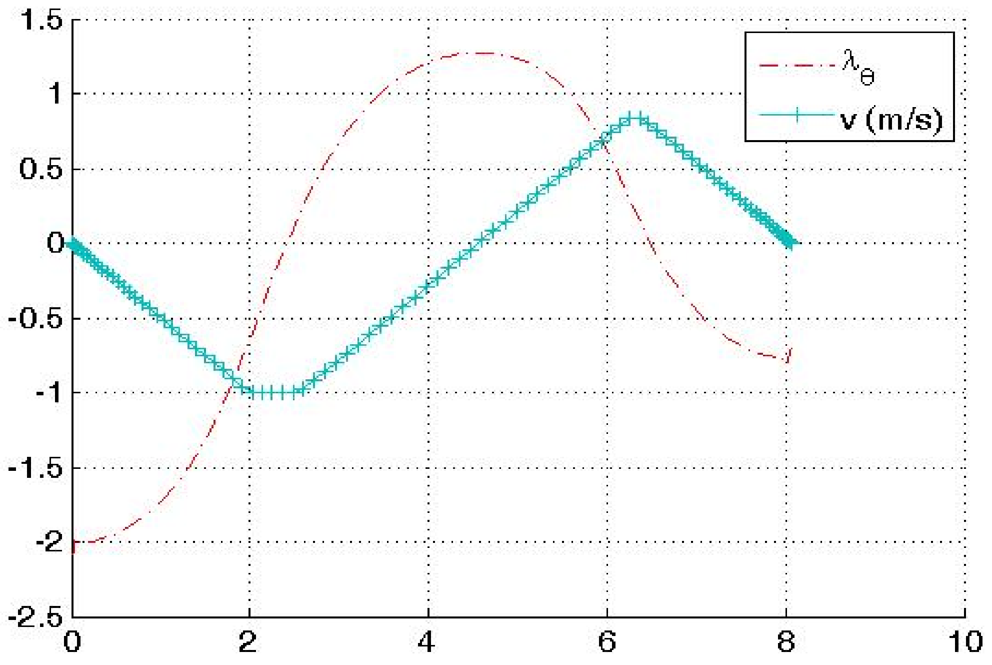


Figure 33. Relationship between λ_θ and *v*.

The next step is to evaluate the Endpoint Lagrangian ($\bar{E}$), which becomes:

$$\begin{aligned}\bar{E} &= t_f + \nu_{t_0} t_0 + \nu_{x_0}\left(x(t_0)-5\right)+\nu_{y_0}\left(y(t_0)-5\right)+\nu_{\theta_0}\theta(t_0)+\nu_{v_0}v(t_0)+\nu_{\varphi_0}\varphi(t_0)+\dots\\ &\dots+\nu_{x_f}\left(x(t_f)-5\right)+\nu_{y_f}\left(y(t_f)-4\right)+\nu_{\theta_f}\theta(t_f)+\nu_{v_f}v(t_f)+\nu_{\varphi_f}\varphi(t_f)\end{aligned} \tag{26}$$

The Endpoint Lagrangian is used to establish the Transversality Conditions:

$$\begin{aligned}&\lambda_x(t_0)=-\nu_{x_0}, \lambda_x(t_f)=\nu_{x_f}\\ &\lambda_y(t_0)=-\nu_{y_0}, \lambda_y(t_f)=\nu_{y_f}\\ &\lambda_\theta(t_0)=-\nu_{\theta_0}, \lambda_\theta(t_f)=\nu_{\theta_f}\\ &\lambda_v(t_0)=-\nu_{v_0}, \lambda_v(t_f)=\nu_{v_f}\\ &\lambda_\varphi(t_0)=-\nu_{\varphi_0}, \lambda_\varphi(t_f)=\nu_{\varphi_f}\end{aligned} \tag{27}$$

Nothing new can be interpreted from the Transversality Condition. The Hamiltonian Value Condition is given by:

$$H(t_0)=\nu_{t_0}, H(t_f)=-1 \tag{28}$$

Combining this result with the Hamiltonian Evolution Equation results in $H=-1$ throughout the vehicle's maneuver. Figure 34 validates this conclusion.

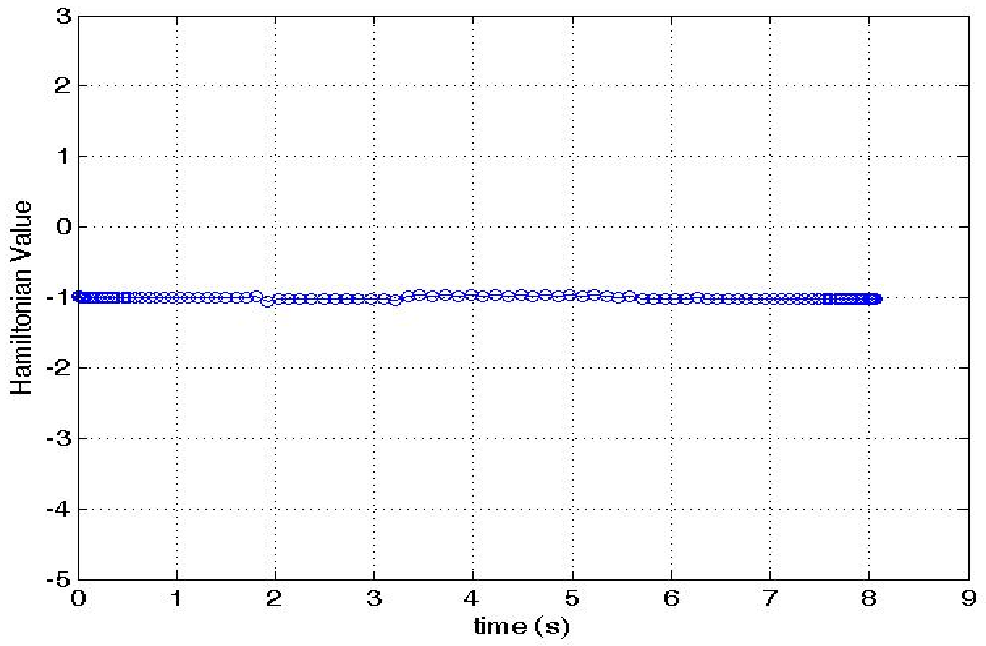


Figure 34. Hamiltonian.

The final step in determining the current extremal as the locally optimal solution is to verify Bellman's Principle. The original trajectory was split up near the 4 second point and the original problem was split into two problems, a first half problem and a second half problem. The two halves were then combined to show an overall trajectory. This overall trajectory was then compared to the original trajectory. Figure 35 shows the x vs y trajectory of the original problem with the 1st half and 2nd half problems superimposed on the same graph; and it can be seen that the pieces add up to the whole. The same holds true for the remaining states θ, v, φ as shown in Figure 36. The cost of the 1st half problem was 3.97; the cost of the 2nd half problem was 4.10; and adding the 1st and 2nd half costs comes out to 8.07, which exactly matches the original problem cost.

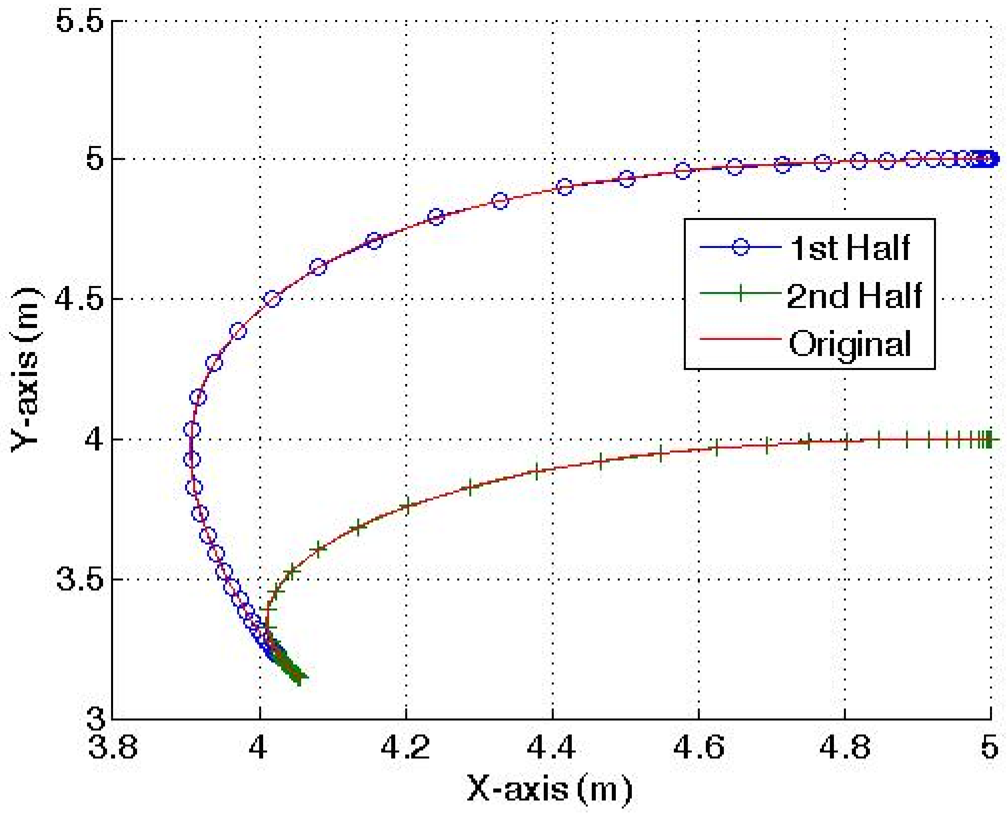


Figure 35. Bellman's Principle (x vs y).

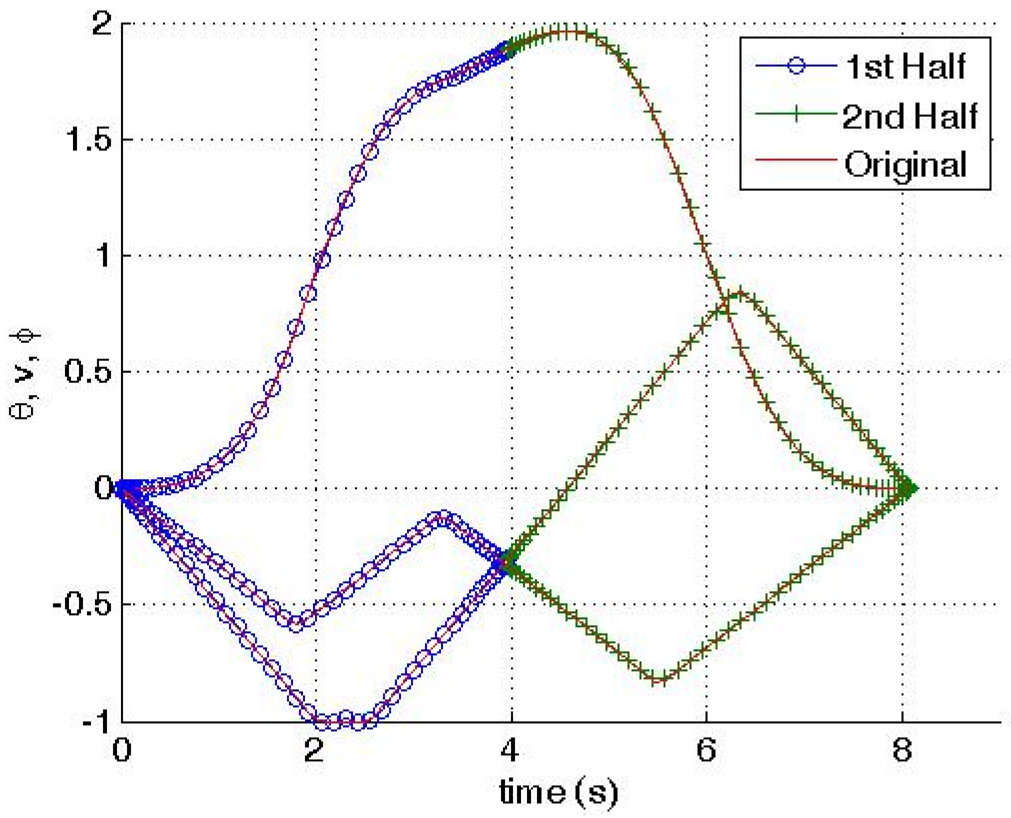


Figure 36. Bellman's Principle (θ, v, φ).

The verification and validation steps performed in this chapter will be performed throughout this work, on all problems and scenarios, so as to maintain the validity of the DIDO generated optimal solutions. However, the verification and validation results will not be displayed unless they offer additional information that is useful in the presentation of this research.

D. FOUR-WHEEL CAR PROBLEM OBSERVATIONS

1. Vehicle Orientations and Endpoint Constraints

It is worth discussing here how the vehicle's orientation at the start and finish of a maneuver can affect the planned trajectory. Figure 37 shows the trajectory the vehicle would follow if the desired maneuver was to simply have the vehicle turn around. In this case the vehicle's initial and final conditions are:

$$\underline{x}_0 = \begin{bmatrix} x_0 \\ y_0 \\ \theta_0 \\ v_0 \\ \varphi_0 \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \text{and} \quad \underline{x}_f = \begin{bmatrix} x_f \\ y_f \\ \theta_f \\ v_f \\ \varphi_f \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \\ \pi \\ 0 \\ 0 \end{bmatrix} \tag{29}$$

The problem was solved using a 50 node solution and the maneuver time (cost) was 11.48 *sec*. The trajectory can be flipped by requiring $\theta_f = -\pi$, shown in Figure 38 without the vehicle drawn on top.

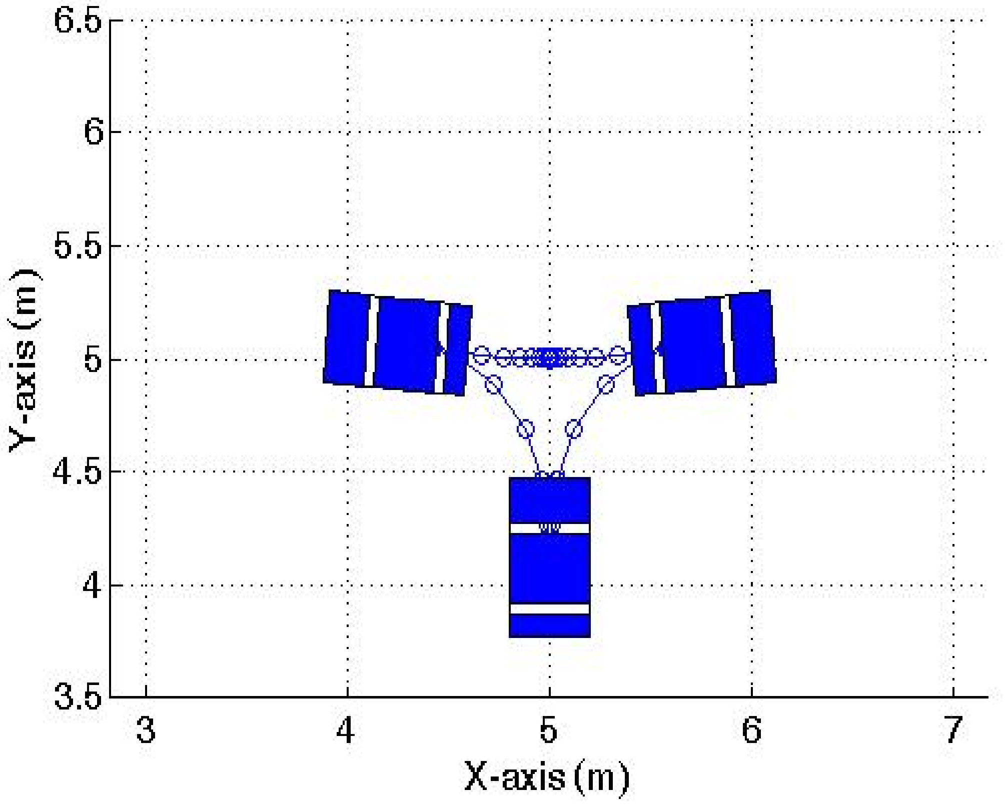


Figure 37. Vehicle Turn Around ($\theta_f = \pi$).

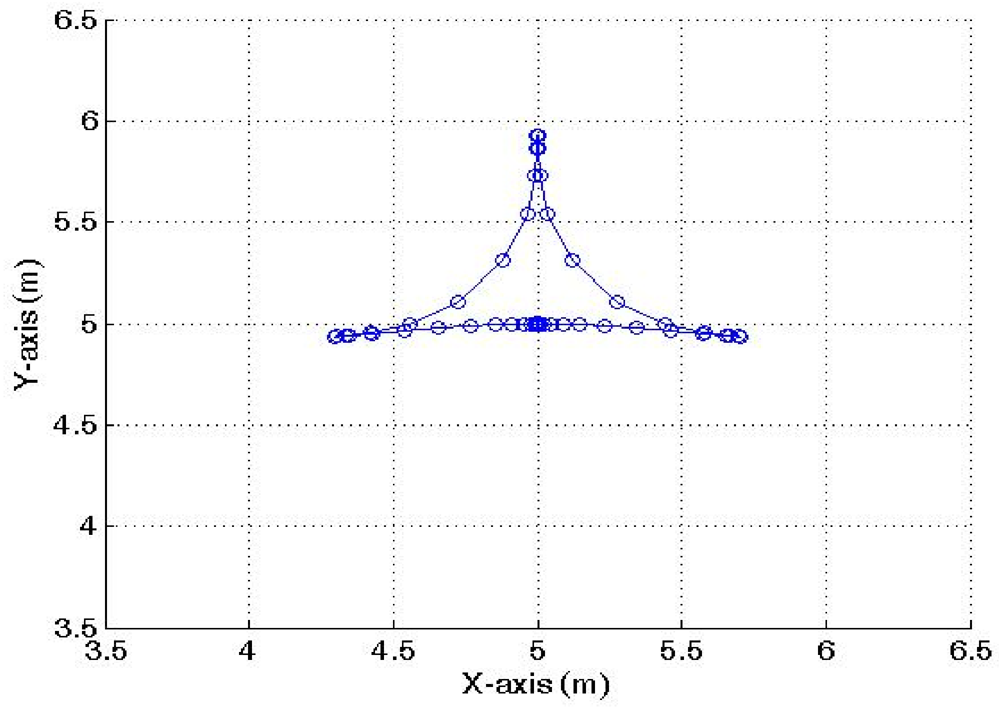


Figure 38. Vehicle Turn Around ($\theta_f = -\pi$).

Figure 39 shows a completely different trajectory that solves the same task. The vehicle starts out at $\theta_0 = 0$ and then turns around. In this case, the endpoint constraints on θ are as follows:

$$\begin{aligned} \sin(\theta_f) &= 0 \\ \cos(\theta_f) &= -1 \end{aligned} \tag{30}$$

The endpoint constraints of equation set (30) allow θ_f to be any value within the limits established on θ in the problem, as long as it meets the requirements of equation set (30). This problem was again solved with 50 nodes and the maneuver time was actually slightly faster at 11.11 *sec*. It is interesting to note here that this solution results in $\theta_f = 3\pi$, which is not what one might intuitively expect. One would expect the vehicle to go to $\theta_f = \pi$ or $\theta_f = -\pi$, but here it finds a slightly less costly maneuver by going to $\theta_f = 3\pi$.

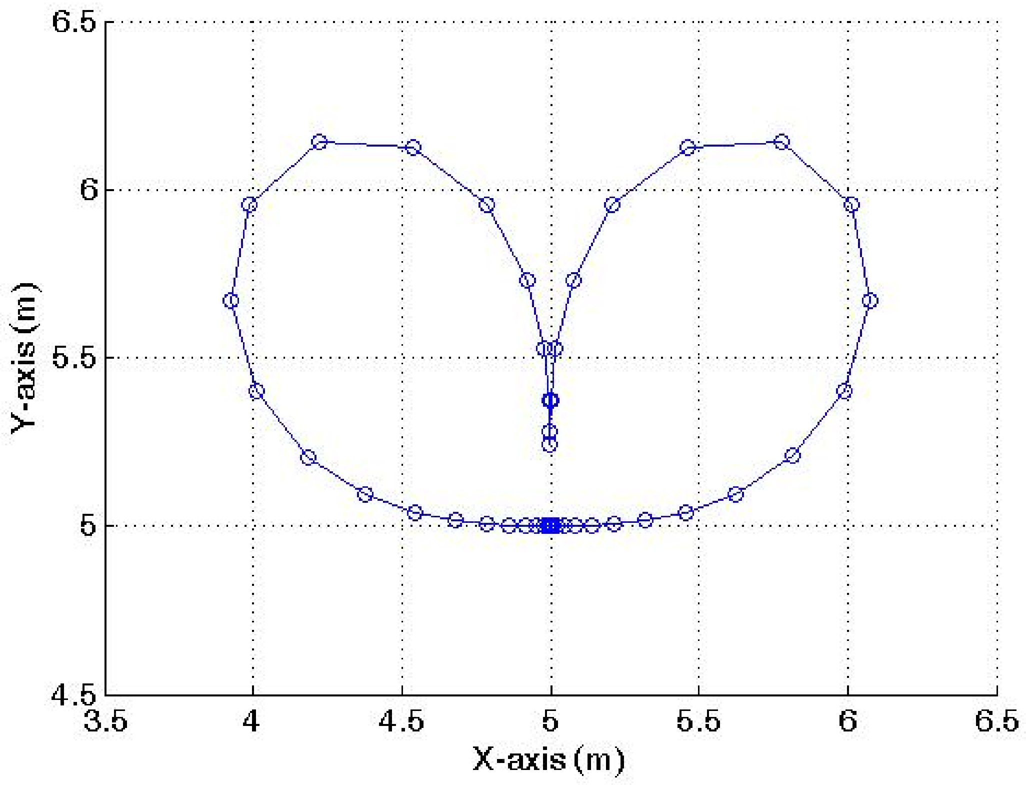


Figure 39. Vehicle Turn Around ($\sin(\theta_f) = 0, \cos(\theta_f) = -1$).

The original vehicle turn around maneuver was conducted again, but this time the final vehicle orientation was defined by the endpoint condition of $\theta_f = 3\pi$. This problem formulation resulted in exactly the same trajectory as that of Figure 39.

The 2nd maneuver to discuss in this section has the following initial and final conditions:

$$\underline{x}_0 = \begin{bmatrix} x_0 \\ y_0 \\ \theta_0 \\ v_0 \\ \varphi_0 \end{bmatrix} = \begin{bmatrix} 5 \\ 15 \\ \pi \\ 0 \\ 0 \end{bmatrix} \quad \text{and} \quad \underline{x}_f = \begin{bmatrix} x_f \\ y_f \\ \theta_f \\ v_f \\ \varphi_f \end{bmatrix} = \begin{bmatrix} 15 \\ 5 \\ \frac{3\pi}{4} \\ 0 \\ 0 \end{bmatrix} \tag{31}$$

A 50 node solution was used and the maneuver time was 16.29 *sec*. The trajectory is shown in Figure 40.

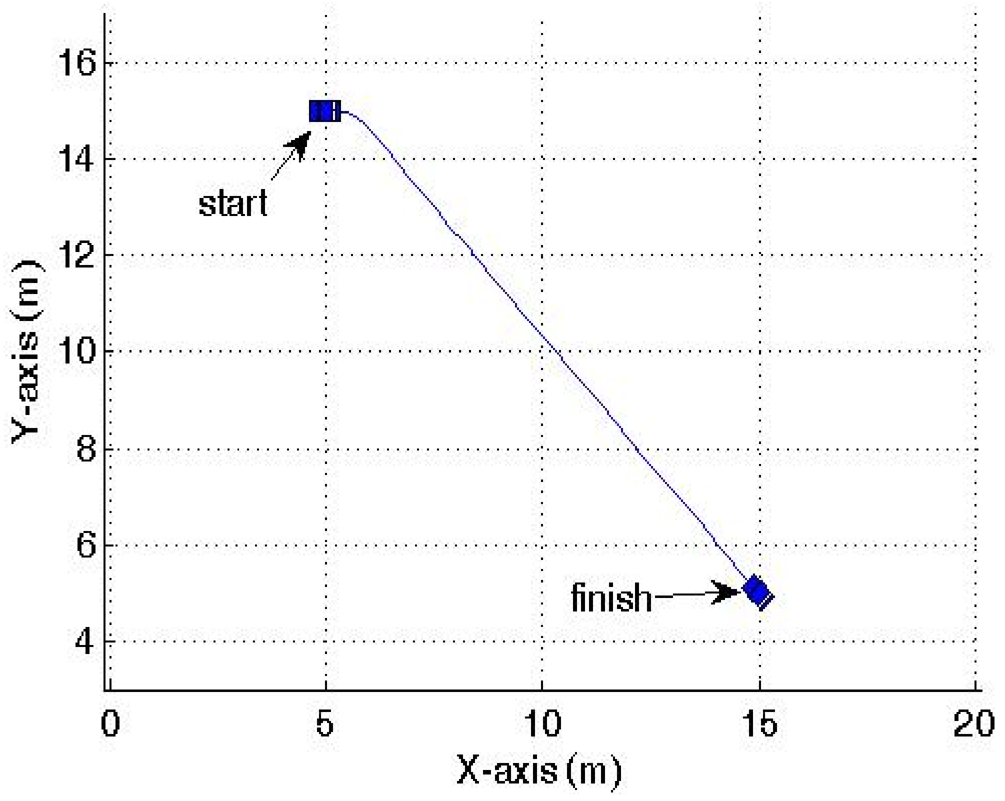


Figure 40. 2nd Maneuver with $\theta_0 = \pi, \theta_f = \frac{3\pi}{4}$.

Figure 41 shows a completely different trajectory to perform the same maneuver. This new trajectory, with a maneuver time of 21.35 *sec*, was produced by simply changing the value of θ_0 to $\theta_0 = -\pi$, which starts the vehicle in the same orientation, but produces different results. If we also use $\theta_f = \frac{-5\pi}{4}$ as the new final condition to go with the $\theta_0 = -\pi$ initial condition, we can produce the exact trajectory of Figure 40 again.

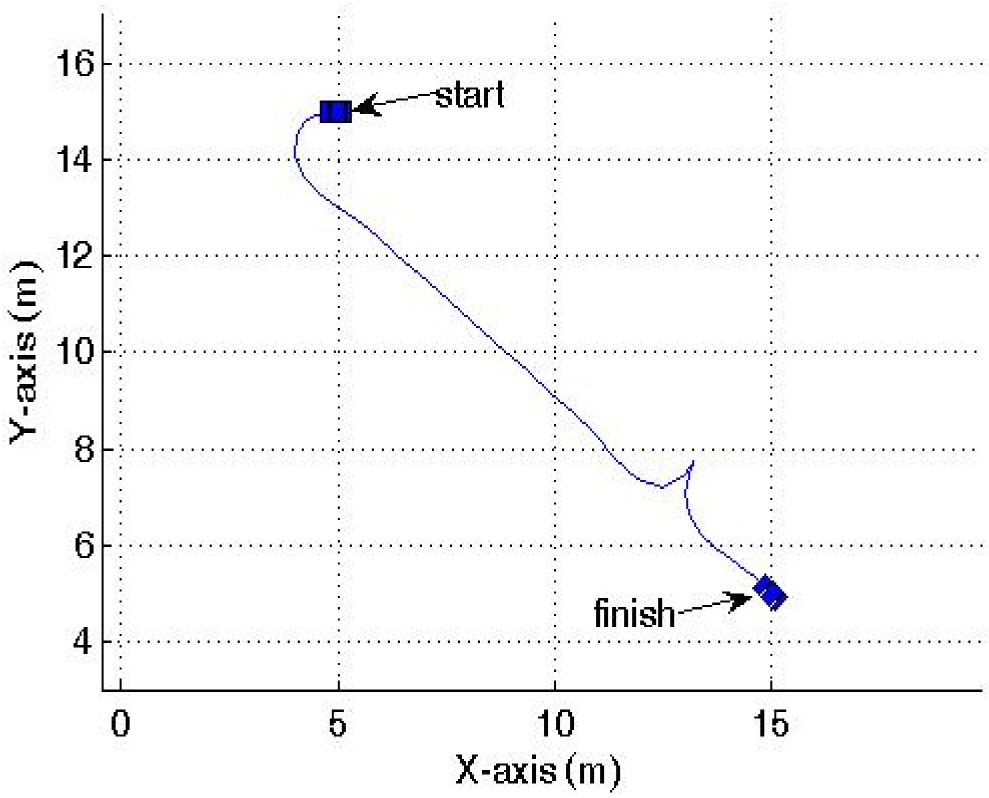


Figure 41. 2nd Maneuver with $\theta_0 = -\pi, \theta_f = \frac{3\pi}{4}$.

The next step was to perform the 2nd maneuver again using $\theta_0 = -\pi$, but this time the final condition corresponds to equation set (32). The resulting trajectory matches that of Figure 40.

$$\begin{aligned}\sin(\theta_f) &= 0.707 \\ \cos(\theta_f) &= -0.707\end{aligned} \tag{32}$$

It can be seen from the above discussions that there is more than one extremal (locally optimal solution) that will solve the problem of maneuvering the vehicle from one orientation/location to another. The solution depends largely on how the vehicle's orientation is defined at the start and finish of its maneuver. There are many different ways to define the vehicle's orientation, and each one can produce a different answer. These concepts must be kept in mind when conducting trajectory planning.

2. Obstacle Avoidance

It was described earlier that all functions and equations used to formulate the optimal control problem must be continuous and differentiable. This translates to the need for describing obstacles with smooth functions. The obstacles themselves need not be smooth, but the functions used to define the obstacle boundaries must be made up of one or more smooth equations. For this reason, obstacle boundaries need not be defined exactly, but can be approximated using one or more simple shapes. The desire here is to have the obstacles clearly defined algebraically.

Most works in the literature are concerned more with the planning method algorithm than with the development of good obstacle functions, therefore, when algebraically defining obstacles they keep it as simple as possible by using circles (in two dimensions) and spheres (in three dimensions). In [18, 21], all obstacles are defined as points with circular danger and avoidance areas. In other words, the boundaries are circular and easy to define. How to model the boundaries of obstacles using polygonal shapes is described in [3]. Let us look at two examples of such a model; a square and a triangle. Equation set (33) describes the boundaries of a square using four straight line equations, and is shown in Figure 42. Equation set (34) and Figure 43 show a triangle.

$$\begin{aligned} x = 0, x = 5 \\ y = 0, y = 5 \end{aligned} \tag{33}$$

$$\begin{aligned} x = 0, y = 0 \\ x + y = 5 \end{aligned} \tag{34}$$

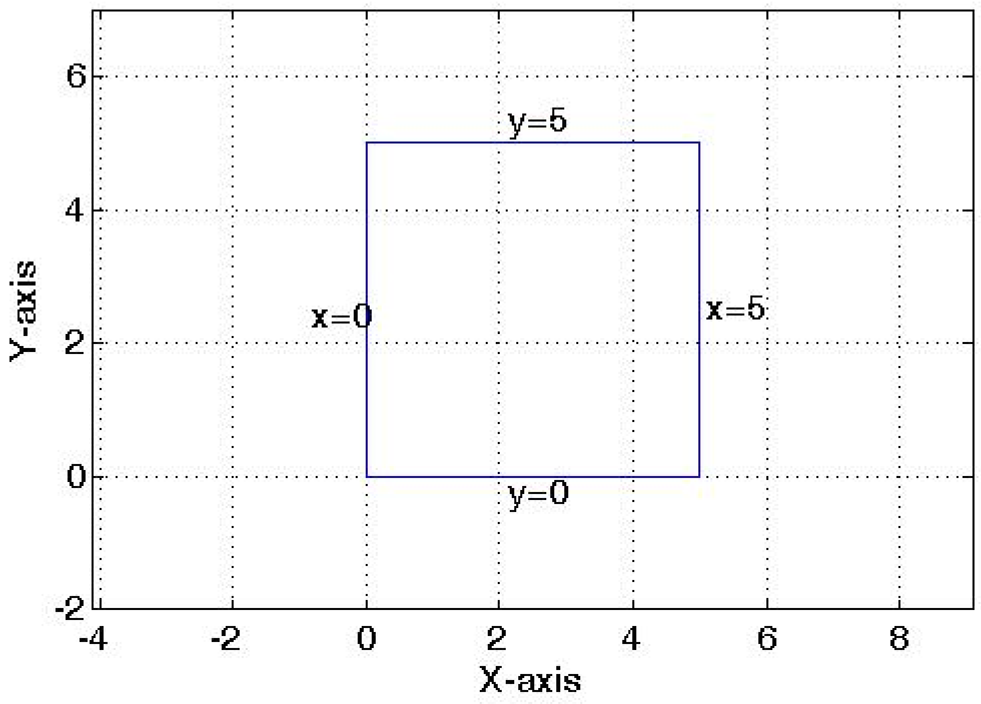


Figure 42. Four Straight Line Equations to Define a Square.

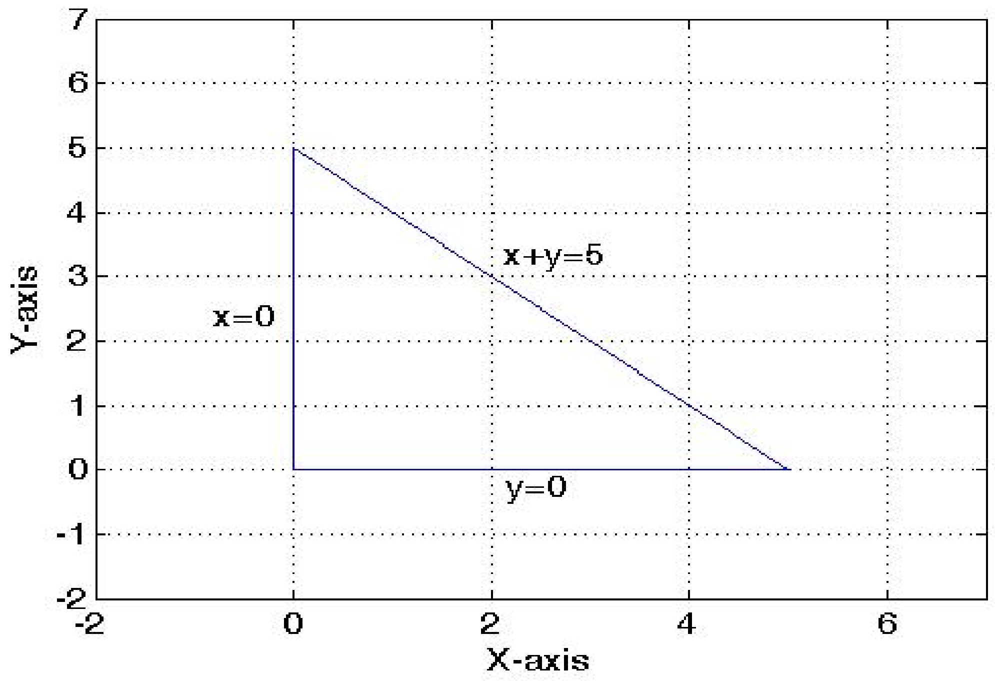


Figure 43. Three Straight Line Equations to Define a Triangle.

The square and triangle shapes defined above are not continuous and differentiable, however, the steps to combine the equations into one smooth function are described in [1, 52].

The obstacle modeling in this work uses the p-norm and was developed in [1, 52, 53, 54]. Using the p-norm, one can easily model any square, rectangle, circle, or ellipse. These shapes are all that is needed in path planning since the boundary of any obstacle can be modeled by fitting one or more of those shapes around it. Modeling an obstacle's exact shape and size is too complex and highly unnecessary. Equation (35) shows the general form of the equation used to model these shapes.

$$h_i(x(t), y(t)) = \left| \left(\frac{x(t) - x_c}{a} \right)^p \right| + \left| \left(\frac{y(t) - y_c}{b} \right)^p \right| - \left| c^p \right| = 0 \tag{35}$$

The variables x_c, y_c indicate the location of the center of the obstacle boundary, while *a*, *b* and *c* are chosen to define the size of the obstacle. We can eliminate the absolute value by limiting the values of *p* to only even numbers, resulting in a continuous and differentiable function. Figure 44 shows examples of the various shapes that can be generated just by varying the values of a, b, p. A value of $c = 1$ is used throughout this work.

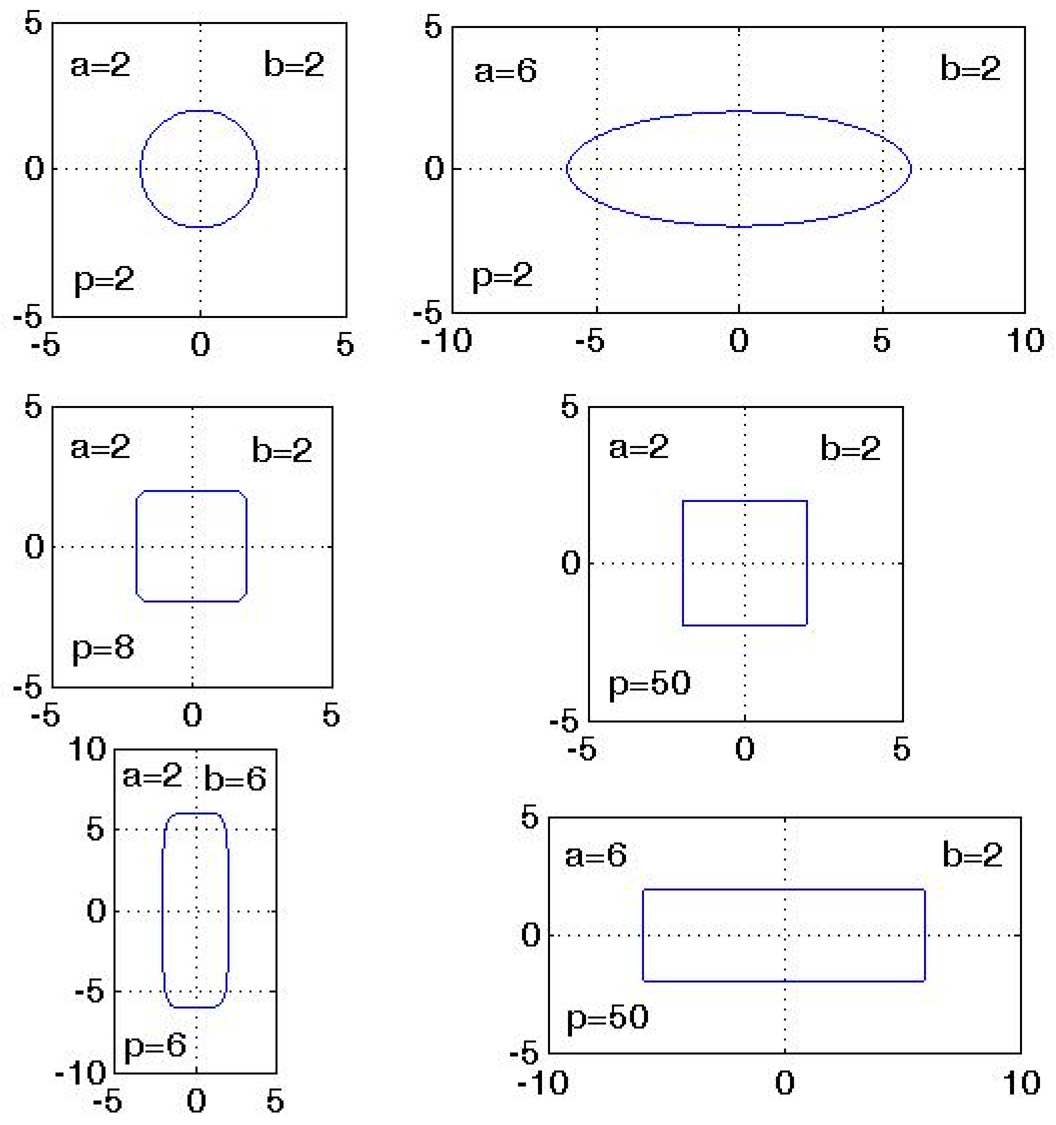


Figure 44. Various Shapes Generated from Equation (35).

An example obstacle avoidance problem is shown in Figure 45. The equation for the obstacle boundary is:

$$h_1(x(t), y(t)) = \left(\frac{x(t)-10}{2}\right)^8 + \left(\frac{y(t)-10}{5}\right)^8 - 1 = 0 \tag{36}$$

It follows that $h_1(x(t), y(t)) < 0$ corresponds to the space inside the obstacle and $h_1(x(t), y(t)) > 0$ indicates the area outside the obstacle and therefore is the desired constraint. The problem here is that equation (36) is not well scaled. Maintaining $h_1(x(t), y(t)) > 0$ can produce very large numbers, which throws the problem out of balance and makes it difficult for DIDO to solve. Rearranging equation (36) and taking the natural log of both sides results in equation (37), which is the desired path constraint that fits in the optimal control problem formulation. This scales down the path constraint of the obstacle and balances out the problem formulation.

$$h_1(x(t), y(t)) = \ln\left(\left(\frac{x(t)-10}{2}\right)^8 + \left(\frac{y(t)-10}{5}\right)^8\right) \geq 0 \qquad (37)$$

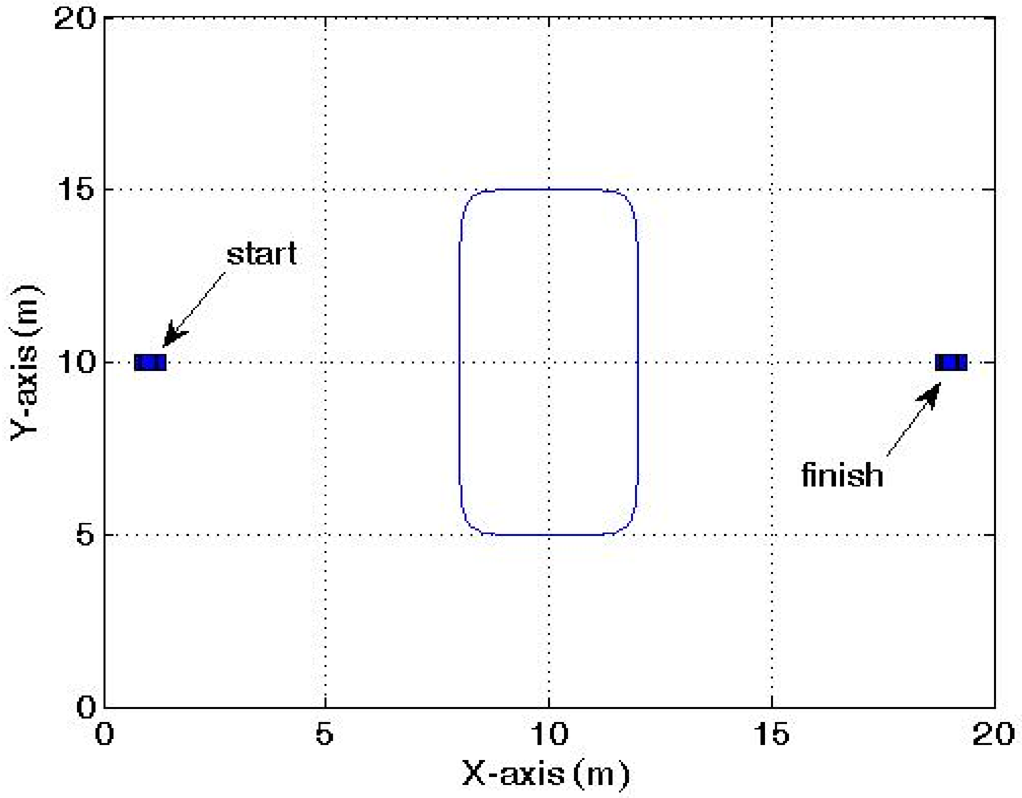


Figure 45. Obstacle Avoidance Problem Setup.

The obstacle avoidance problem of Figure 45 was solved using a 60 node solution and is shown in Figure 46. The overall maneuver time (cost) was 23.13 *sec*.

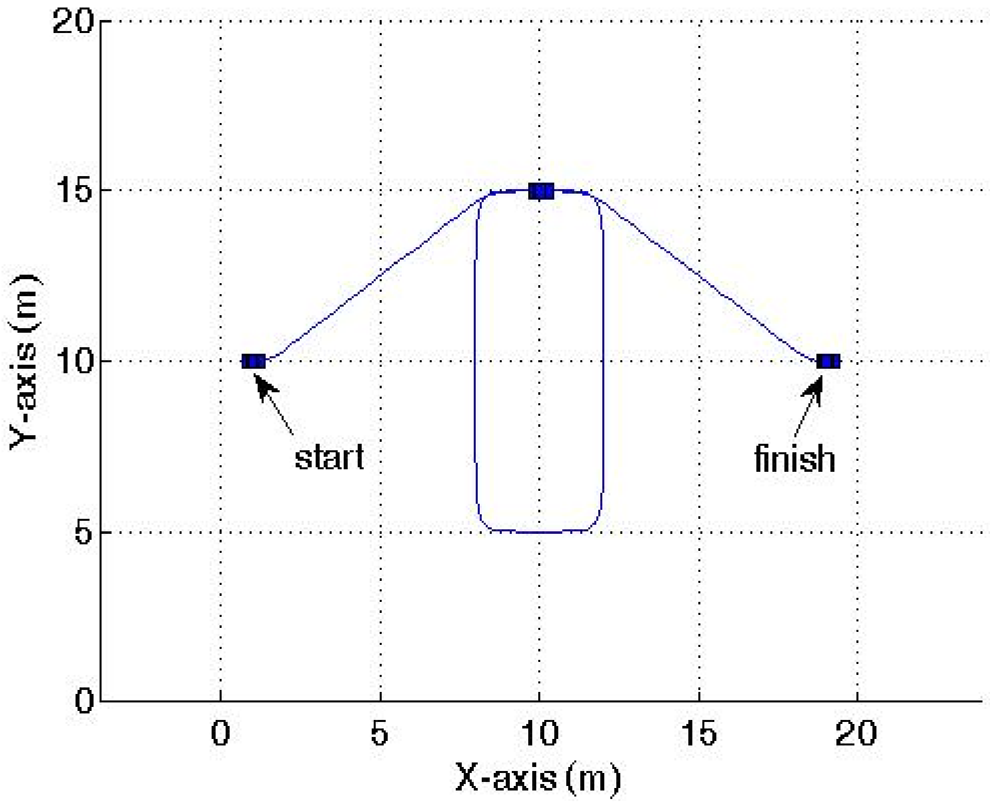


Figure 46. Obstacle Avoidance Problem.

The path constraint of the obstacle in this problem does not change the Hamiltonian of equation (22), but it does add a term, $\mu_{h_1} h_1$, to the Lagrangian of the Hamiltonian in equation (23). The problem has state and control covectors as before, but now it also has a path covector, μ_{h_1}. The KKT conditions require $\mu_{h_1} \leq 0$ when the vehicle is touching the obstacle, i.e., $h_1(x(t), y(t)) = 0$; and $\mu_{h_1} = 0$ when not touching the obstacle. This relationship is shown in Figure 47. It also follows from the Adjoint Equations that λ_x, λ_y are constant when not touching the obstacle, but can change when touching the obstacle, as shown in Figure 48. There are no other significant changes from previous formulations that would provide any more useful interpretations. The trajectory shown in Figure 46 is the locally optimal solution since it meets Pontryagin's Minimum Principle, and it satisfies Bellman's Principle and feasibility.

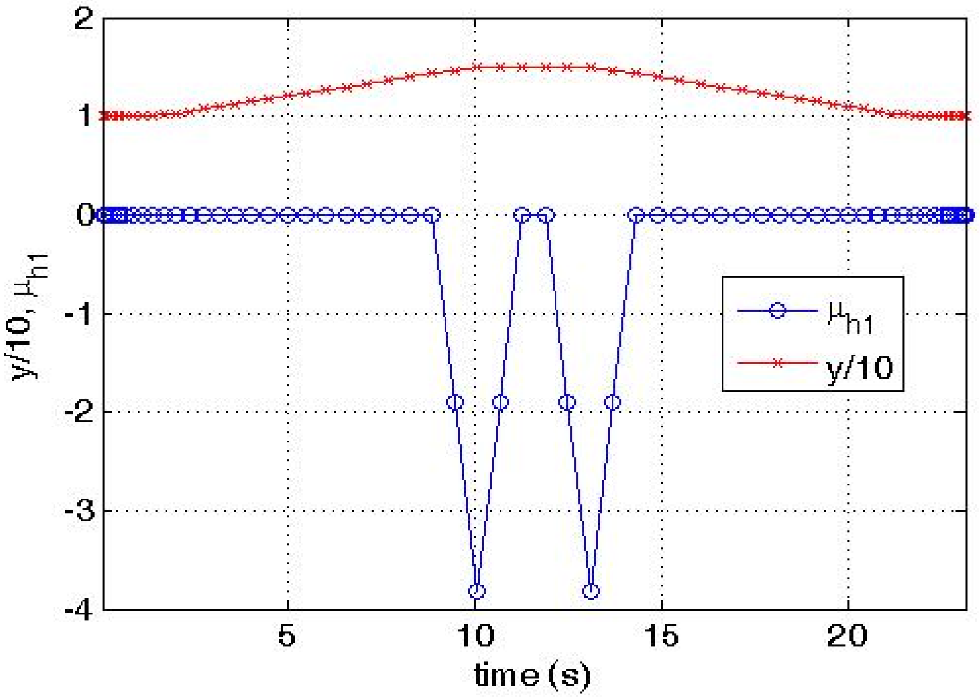


Figure 47. Relationship between μ_{h_1} and y.

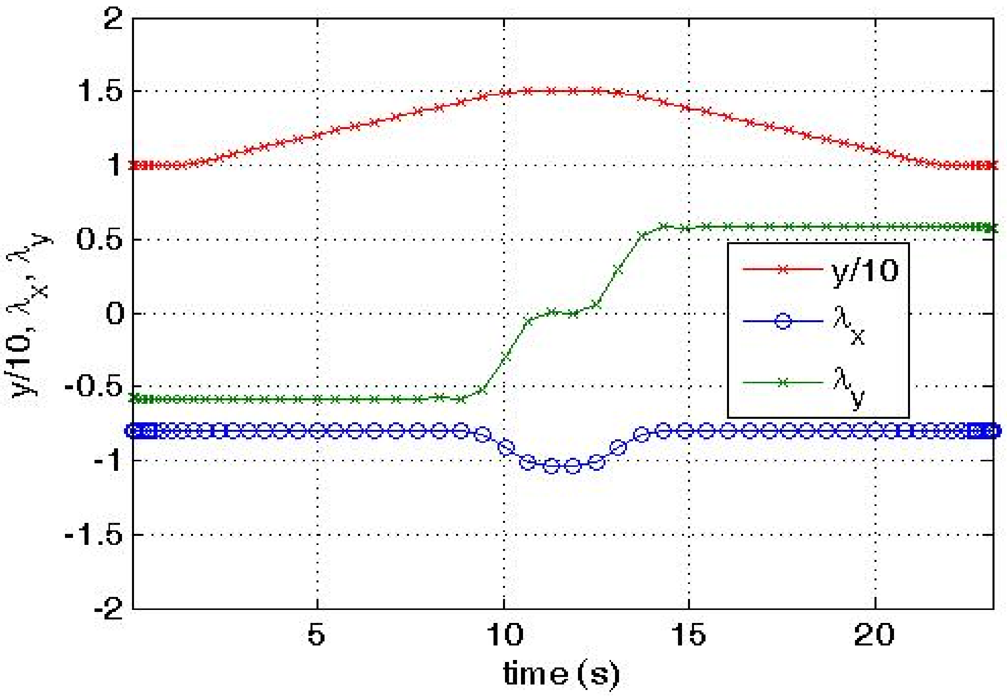


Figure 48. Relationship between λ_x, λ_y and y.

The only problem with the obstacle avoidance solution discussed above is the fact that the vehicle touches the obstacle. This is due to the fact that the solution allows for the touching of obstacles (the equality in the constraint), and because it is a point solution and does not take into account the size of the vehicle. In order to clear the obstacle and avoid contact, a buffer must be added to the path constraint which effectively expands the obstacle boundary by an amount that accounts for the size of the vehicle and will thereby not allow the vehicle to come in contact with the obstacle. For future scenarios, all path constraints will have a *0.5m* buffer (accounting for the size of the four-wheel car being used in this work) added around the outside edge of the obstacle boundary. For the obstacle avoidance problem in this section, equation (37) becomes:

$$h_1(x(t), y(t)) = \ln\left(\left(\frac{x(t)-10}{2.5}\right)^8 + \left(\frac{y(t)-10}{5.5}\right)^8\right) \geq 0 \qquad (38)$$

Equation (38) defines the *expanded* obstacle, which is used in the formulation and solution of the trajectory planning problem. Figure 49 shows the resulting trajectory (23.85 *sec* maneuver time). The expansion of the obstacle in the problem formulation is shown here as a dotted line around the actual obstacle. In future scenarios, the expanded obstacles will be included as dotted lines (when needed for clarity).

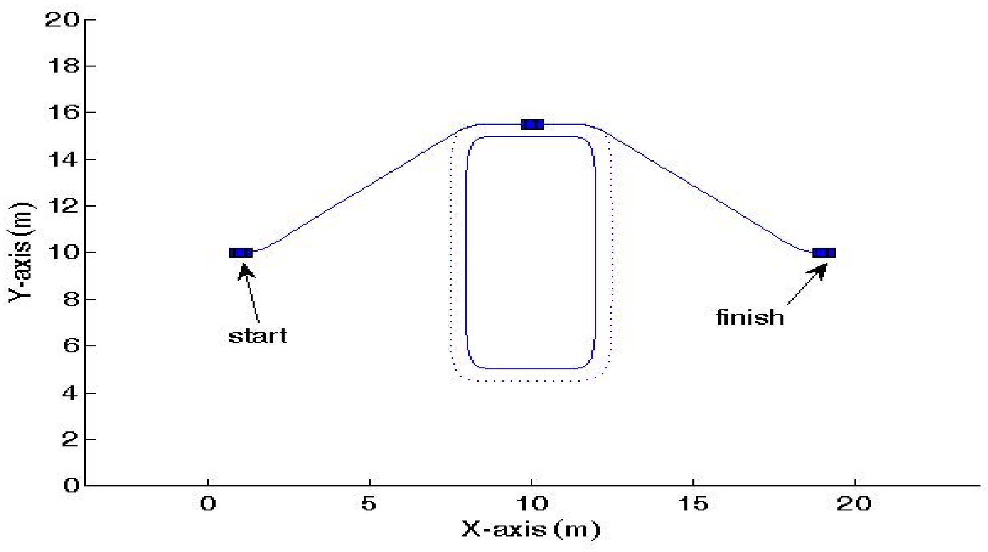


Figure 49. Obstacle Avoidance Problem with *0.5m* buffer.

3. Computational Complexity

The biggest concern in solving Optimal Control problems is the computational complexity, because this can create run times that would preclude the use of this technique in real-time. The more complex the problem is to solve (such as an obstacle rich environment) the longer the run time will be to provide a trajectory for the vehicle. It has already been stated that one of the advantages to providing a bias is that it speeds up the problem formulation considerably from seconds to fractions of a second. It has also been postulated that solution speeds can be increased 100 times (a conservative estimate) simply by optimizing the code, eliminating the Windows and MATLAB overhead [50], advancements in sparse linear algebra, full implementation of spectral algorithms, and improving computer processor speeds [51].

Studying the viability of the Optimal Control method in real-time for a UGV begins by keeping two concepts in mind. First, for a relatively slow UGV (max speed of *1 m/s*), it has been proposed that run times of less than 0.5 seconds would be sufficient for real-time operation. Second, with the advancements and improvements proposed above, a 100 fold increase in the speed of the solutions can be achieved. With these two concepts in mind it is only necessary to show that run times can be kept under 50 seconds. As stated in Chapter II.D, the purpose of this work is not to prove the technique can be run in real time. Calculation speeds will only be used to compare the various scenarios and methods for improving speed. It can never be claimed that fast enough calculation speeds have been reached, because there will always be a demand for more complex missions that will ultimately require faster speeds. The important thing here is understanding the techniques for speeding up the solution process. For information purposes, all simulations in this work were conducted using MATLAB on an Intel(R) Core(TM)2 Quad processor at 2.46 GHz with 2.96 GB of RAM, running Microsoft Windows XP Professional.

It is important to remember that no vehicle has to be sent out on a mission cold, with no idea what trajectory to follow initially. The time before the start of a mission can be used to calculate an initial trajectory with no limits on the run time. This initial trajectory would not have a bias to speed up its solution, nor would a bias be necessary.

However, once the vehicle starts its mission, it must be able to autonomously update its trajectory in real-time. The initial trajectory (solved offline) can be used as a bias for the next calculation (solved online), and each subsequent solution can be used to bias the next solution. By biasing the solutions, the run times can be brought down to the desired level, allowing for continued real-time operation. As a preliminary example, we look at the solution of Figure 49. The run time to solve that formulation using 60 nodes with no bias was 34.3 seconds. When the 60 node trajectory is used as a bias, the run time drops to 2.2 seconds, which is more than low enough to facilitate real-time operation. This is greater than a 15x speed-up from the original ‘no bias’ problem.

It would be appropriate here to show DIDO’s [56] progression at generating extremals for properly formulated Optimal Control problems. The point here is to say that in solving the obstacle maneuver of Figure 49, the run time of 34.3 seconds is very good considering there is no bias given in the formulation. The problem must be solved with no help. Figures 50, 51, 52, 53, 54 show the progression of the 60 node solution using no bias. The entire solution comprises 23,340 iterations in just 34.3 seconds. The Figures label each trajectory by the iteration number that DIDO was currently on. It can be seen that a collision-free path already existed by the 5000th iteration (about 7.4 seconds of run time). The remaining iterations are spent refining the solution. By the 15,000th iteration (22.1 seconds of run time), the solution is nearly identical to the final result. This shows how well DIDO can hone in on the optimal trajectory with no help at all. When using the 60 node trajectory as a bias, the 2.2 second solution time uses only 1269 iterations. So, why is it so important to lower the run time using a bias when the 60 node open loop run time is almost fast enough already for real-time operation? Remember that the problem presented in Figure 49 is simple and covers a small area. As the area of operation becomes larger (such as a *20km* square grid vice a *20m* grid) and the complexity of the scenario increases, algorithm run times will become longer and longer, and reducing run times becomes paramount to the successful real-time operation of this trajectory planning technique. Another way of lowering algorithm run times is by using the lowest number of nodes possible. Could the maneuver of Figure 49 be solved with 15 node solutions? If so, how would that improve the run time? These questions will be answered later in this work.

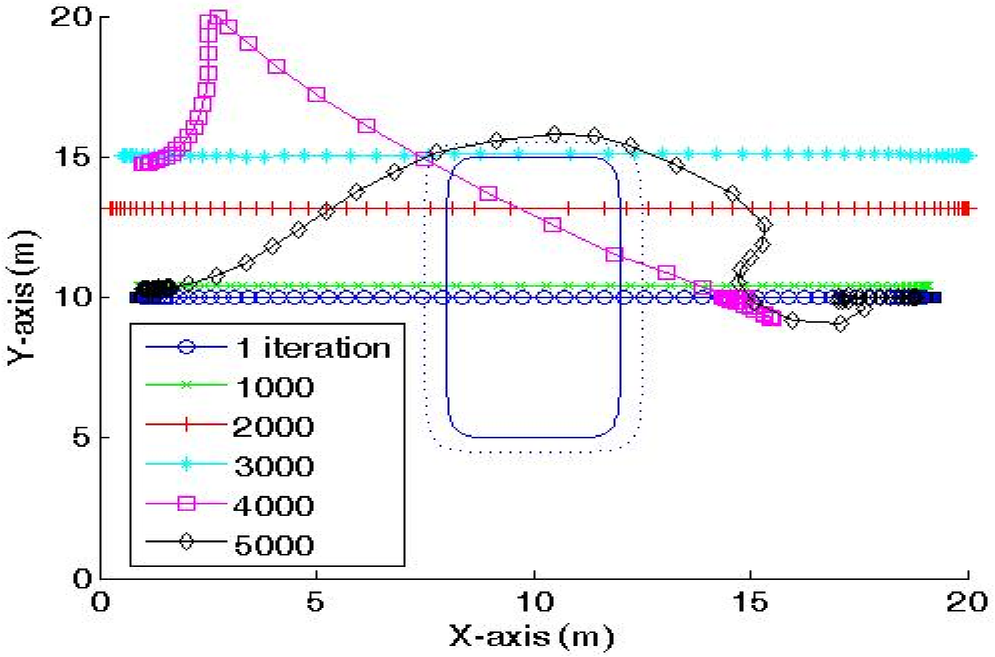


Figure 50. Solution Progression (First 5000 Iterations).

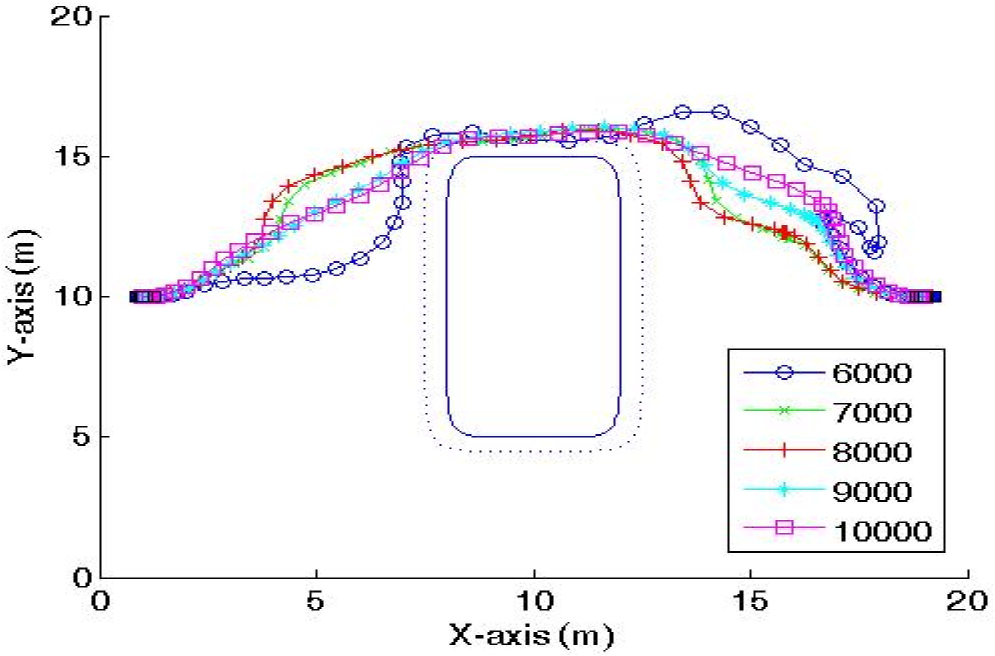


Figure 51. Solution Progression (6000-10,000 Iterations).

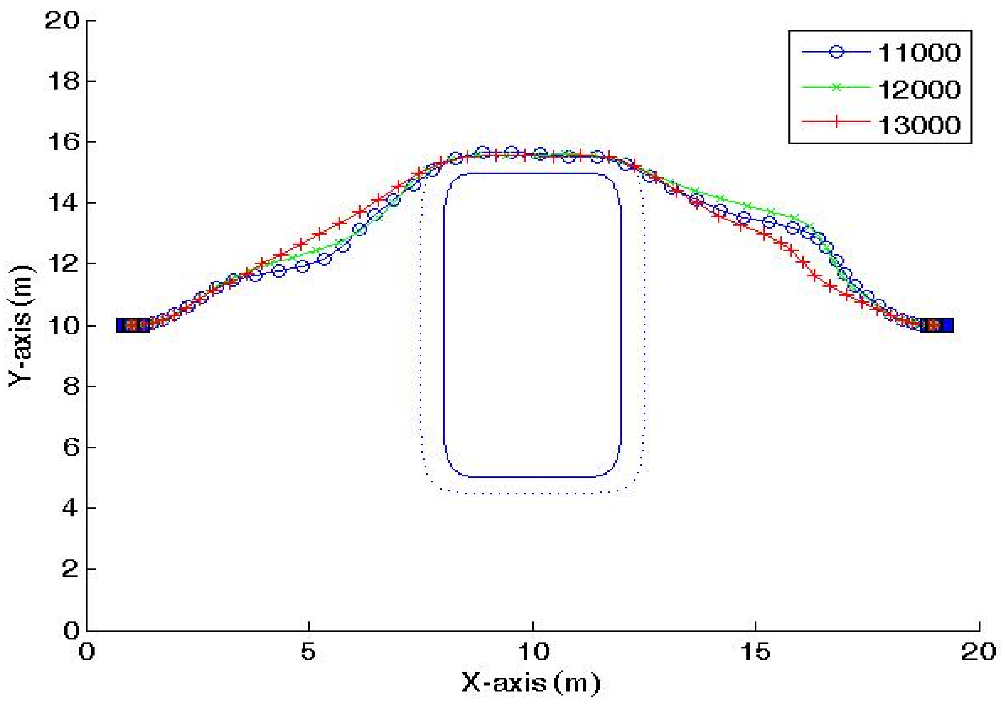


Figure 52. Solution Progression (11,000-13,000 Iterations).

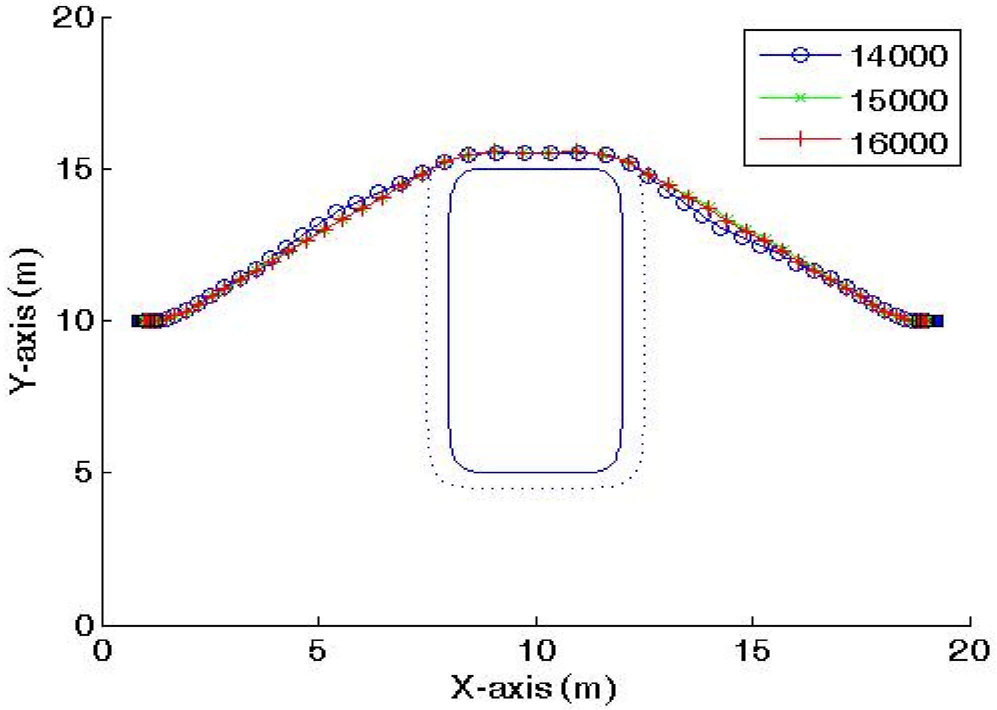


Figure 53. Solution Progression (14,000-16,000 Iterations).

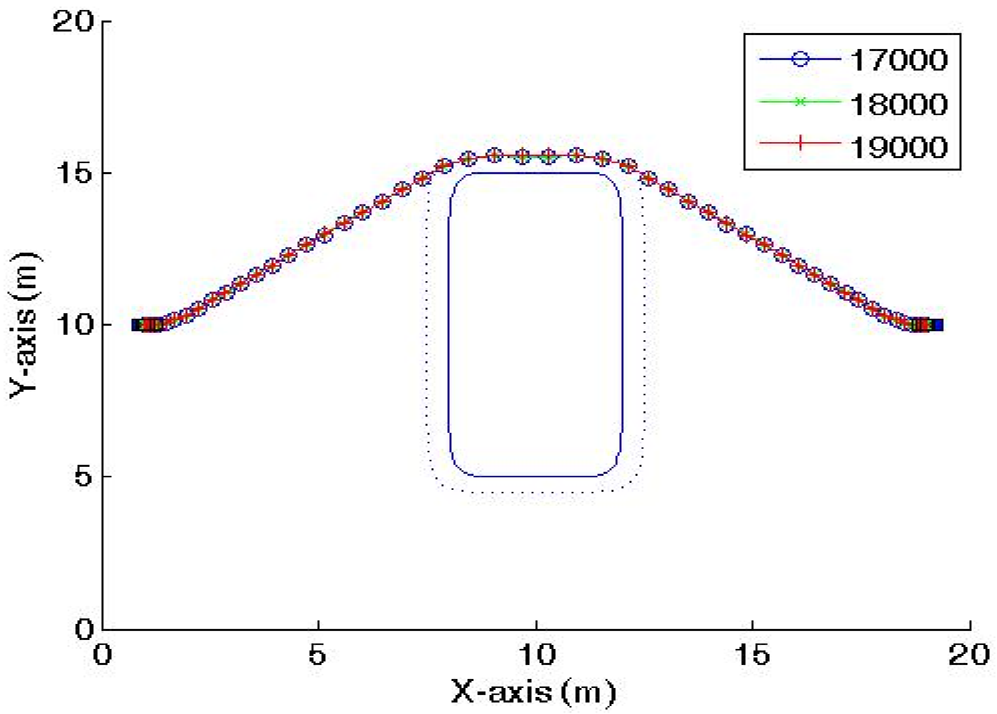


Figure 54. Solution Progression (17,000-19,000 Iterations).

The next step is to show how a bias speeds up the solution to a more complicated problem. The obstacle rich environment of Figure 55 was chosen as the scenario to be solved. The vehicle's initial and final conditions were as follows:

$$\underline{x}_0 = \begin{bmatrix} x_0 \\ y_0 \\ \theta_0 \\ v_0 \\ \varphi_0 \end{bmatrix} = \begin{bmatrix} 1 \\ 10 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \text{and} \quad \underline{x}_f = \begin{bmatrix} x_f \\ y_f \\ \theta_f \\ v_f \\ \varphi_f \end{bmatrix} = \begin{bmatrix} 18.5 \\ 5 \\ \frac{\pi}{2} \\ 0 \\ 0 \end{bmatrix} \tag{39}$$

A 60 node solution was obtained with no bias and is shown as Figure 56. The run time for this calculation was 37.9 seconds. The resulting trajectory in Figure 56 was used as the bias for another 60 node run, which resulted in a run time of 2.5 seconds. Again, it can be seen that using a bias speeds up the run time to well within the standard for real-time (online) operation.

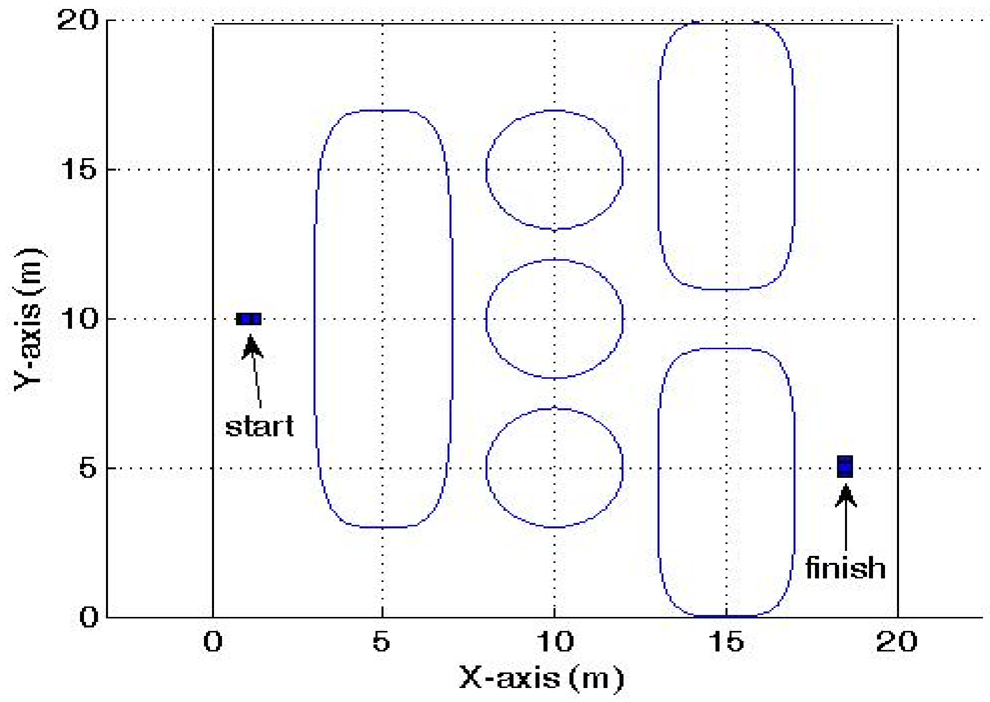


Figure 55. Obstacle Rich Scenario.

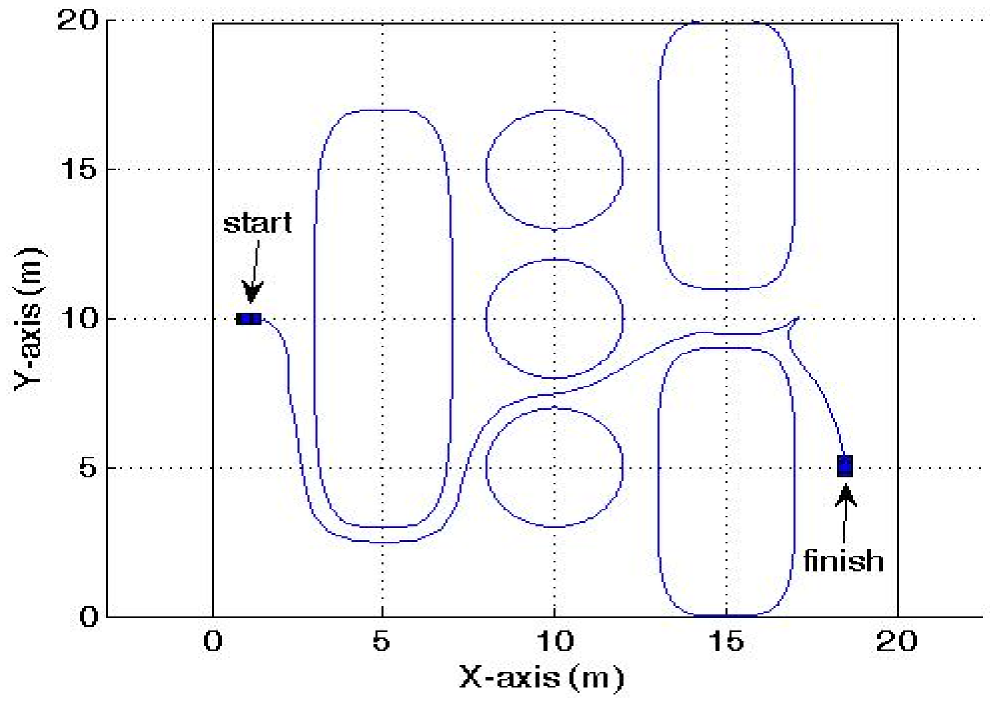


Figure 56. Obstacle Rich Trajectory.

E. AN ALTERNATE PROBLEM FORMULATION

The purpose of this section is to show that there is not just one way to formulate the Optimal Control problem. Any conceivable formulation can be used as long as all dynamics and constraints are clearly defined, the functions are smooth (continuous and differentiable), and the equations and variables are well scaled and balanced. The same vehicle (Figure 20) will be used along with the original sideways maneuver of Figure 21. It will be shown that with a different problem formulation the same end result can be achieved.

1. Alternate Four-Wheeled Car Optimal Control Problem

It is proposed here that we desire to model the vehicle without using the θ variable. Keeping in mind the original kinematic equations in equation set (17), we start by making the following assignment:

$$\begin{aligned} I_x &= \cos(\theta) \\ I_y &= \sin(\theta) \end{aligned} \tag{40}$$

This results in new equations for $\dot{x}, \dot{y}$:

$$\begin{aligned} \dot{x} &= vI_x \\ \dot{y} &= vI_y \end{aligned} \tag{41}$$

Taking the time derivatives of equation set (40) and using the fact that $\dot{\theta} = 2v\tan(\varphi)$ results in:

$$\begin{aligned} \dot{I}_x &= -2vI_y \tan(\varphi) \\ \dot{I}_y &= 2vI_x \tan(\varphi) \end{aligned} \tag{42}$$

The above transformations result in the following Optimal Control problem statement:

Given $$\underline{x} = \begin{bmatrix} x \\ y \\ I_x \\ I_y \\ v \\ \varphi \end{bmatrix} \in X \subset \mathbb{R}^6 \qquad \underline{u} = \begin{bmatrix} a \\ \omega \end{bmatrix} \in U \subset \mathbb{R}^2$$

Find $\underline{x}(\cdot), \underline{u}(\cdot)$

That Minimizes $J = t_f$

Subject to $$\dot{\underline{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{I}_x \\ \dot{I}_y \\ \dot{v} \\ \dot{\varphi} \end{bmatrix} = \begin{bmatrix} vI_x \\ vI_y \\ -2vI_y \tan(\varphi) \\ 2vI_x \tan(\varphi) \\ a \\ \omega \end{bmatrix} \tag{43}$$

$$\begin{aligned} &0 \le x(t) \le 10 \\ &0 \le y(t) \le 10 \\ &-1 \le I_x \le 1 \\ &-1 \le I_y \le 1 \\ &-1 \le v(t) \le 1 \\ &-1 \le \varphi(t) \le 1 \\ \\ &-0.5 \le a(t) \le 0.5 \\ &-0.33 \le \omega(t) \le 0.33 \end{aligned} \qquad e(\underline{x}_0, \underline{x}_f, t_0, t_f) = \begin{bmatrix} t_0 \\ x(t_0)-5 \\ y(t_0)-5 \\ I_x(t_0)-1 \\ I_y(t_0) \\ v(t_0) \\ \varphi(t_0) \\ x(t_f)-5 \\ y(t_f)-4 \\ I_x(t_f)-1 \\ I_y(t_f) \\ v(t_f) \\ \varphi(t_f) \end{bmatrix} = [\underline{0}]$$

2. Verification and Validation

The Optimal Control problem of Figure 21 was solved numerically using DIDO [56]. A 100 node solution was obtained, and the overall maneuver time (cost) was *8.19 sec*. Figure 57 shows the complete maneuver, while Figures 58 and 59 show the state and control trajectories respectively. Notice that the maneuver time is slightly longer than in the original problem formulation, and the trajectory is different due to the fact that it solved for a different extremal (not unusual given the fact that it is an alternate problem formulation). But the most important thing to see here is that the end result is the same; the vehicle has moved to its right by *1m*, and the cost was relatively equal.

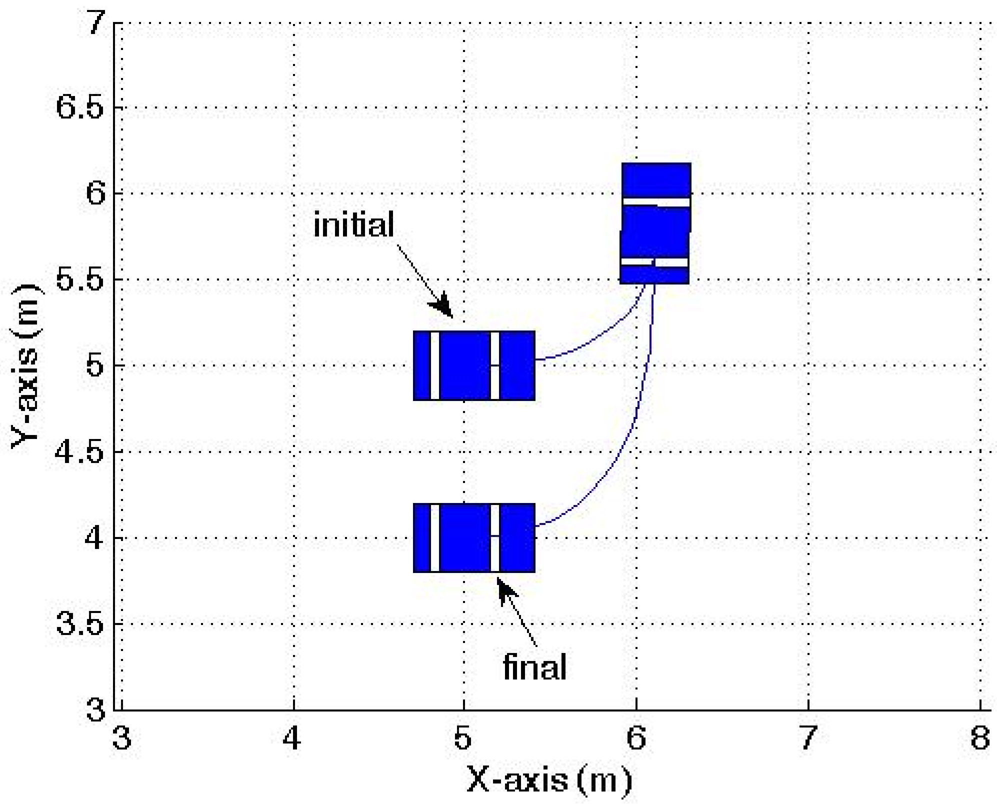


Figure 57. Sideways Maneuver (Alternate Formulation).

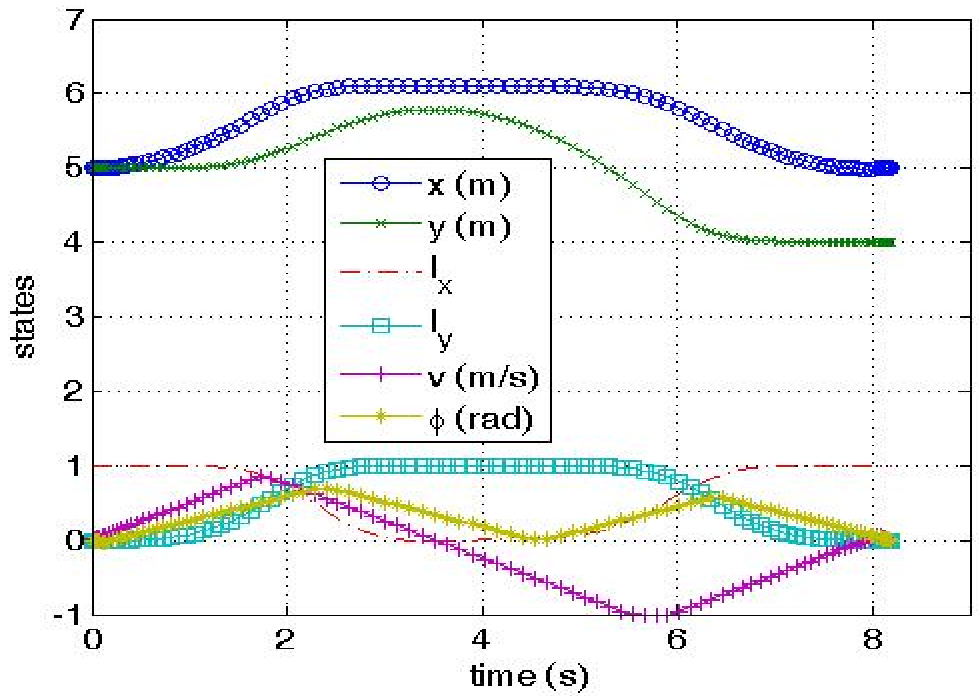


Figure 58. State Trajectory (Alternate Formulation).

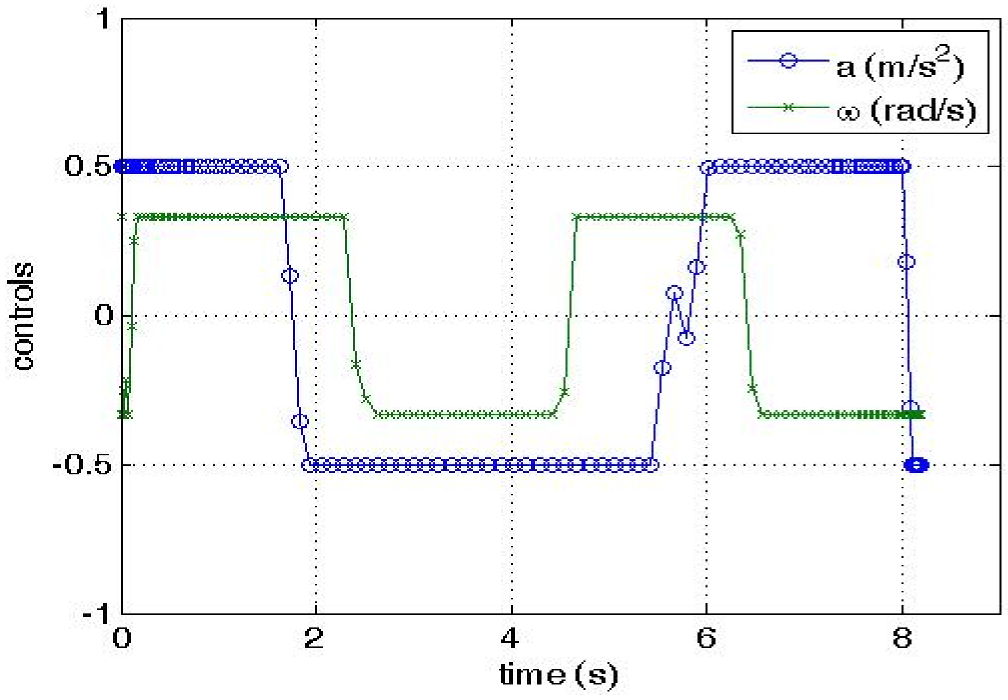


Figure 59. Control Trajectory (Alternate Formulation).

The first step in verification is that of showing the feasibility of the generated solution. The v, φ state variables were used as the control solution. The initial conditions and system dynamics of the x, y, I_x, I_y state variables were propagated using the v, φ control solution. Figures 60 and 61 show the propagated *x* vs *y* and I_x, I_y vs *t* trajectories superimposed on the DIDO solutions, and they are nearly identical, which tells us that the optimal solution is feasible.

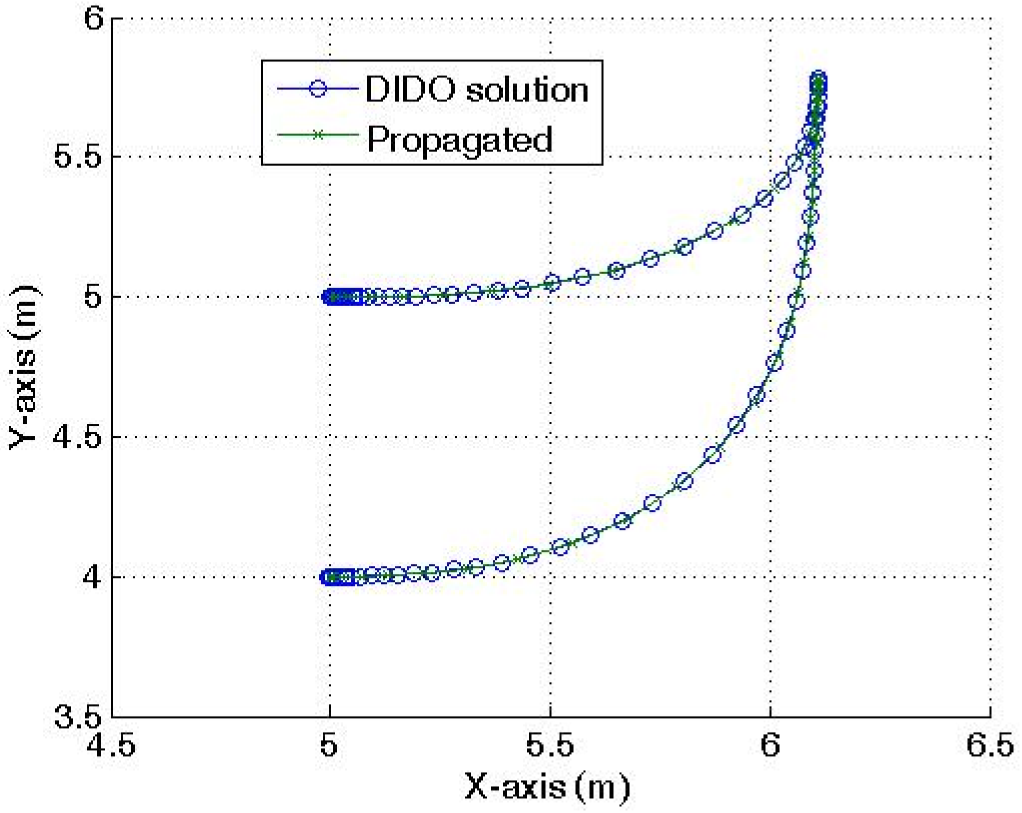


Figure 60. Feasibility Check (*x* vs *y*).

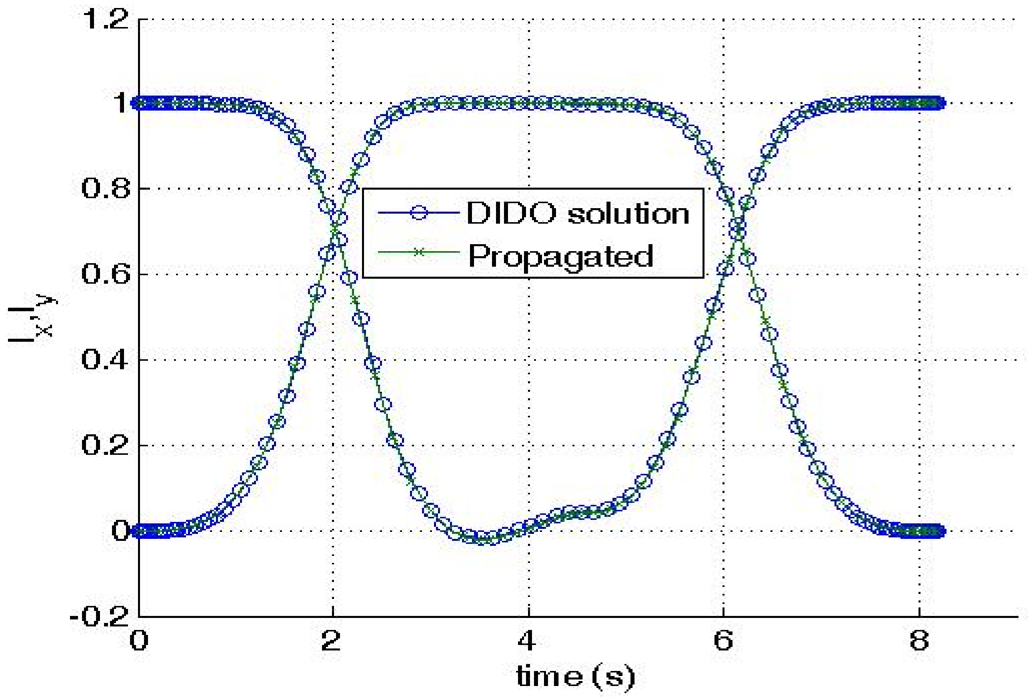


Figure 61. Feasibility Check (I_x, I_y vs *t*).

Verifying Pontryagin's Minimum Principle involves first obtaining the Hamiltonian (*H*):

$$H = \lambda_x v I_x + \lambda_y v I_y - \lambda_{I_x} 2 v I_y \tan(\varphi) + \lambda_{I_y} 2 v I_x \tan(\varphi) + \lambda_v a + \lambda_\varphi \omega \tag{44}$$

The Lagrangian of the Hamiltonian ($\bar{H}$) becomes:

$$\bar{H} = H + \mu_x x + \mu_y y + \mu_{I_x} I_x + \mu_{I_y} I_y + \mu_v v + \mu_\varphi \varphi + \mu_a a + \mu_\omega \omega \tag{45}$$

From Figure 58, it can be seen that x, y, φ stay inside their limits, resulting in their state covectors remaining equal to zero (Figure 62). The KKT conditions require the state covectors $\mu_{I_x}, \mu_{I_y}, \mu_v$ be less than or equal to zero during the time frame that I_x, I_y, v are at their minimum limits, and greater than or equal to zero during the time frame that I_x, I_y, v are at their maximum limits. It just so happens that μ_{I_x} is constant and zero, which is also shown in Figure 62. The relationships between μ_{I_y}, μ_v and I_y, v are shown on Figure 63. Notice that μ_{I_y} is greater than zero only when I_y is at its maximum value and μ_v is less than zero only when v is at its minimum value. It follows when comparing the control covectors to the control variables (Figure 64) that the covectors oscillate between positive and negative as the control variables switch between maximum and minimum values, as again required by the KKT conditions.

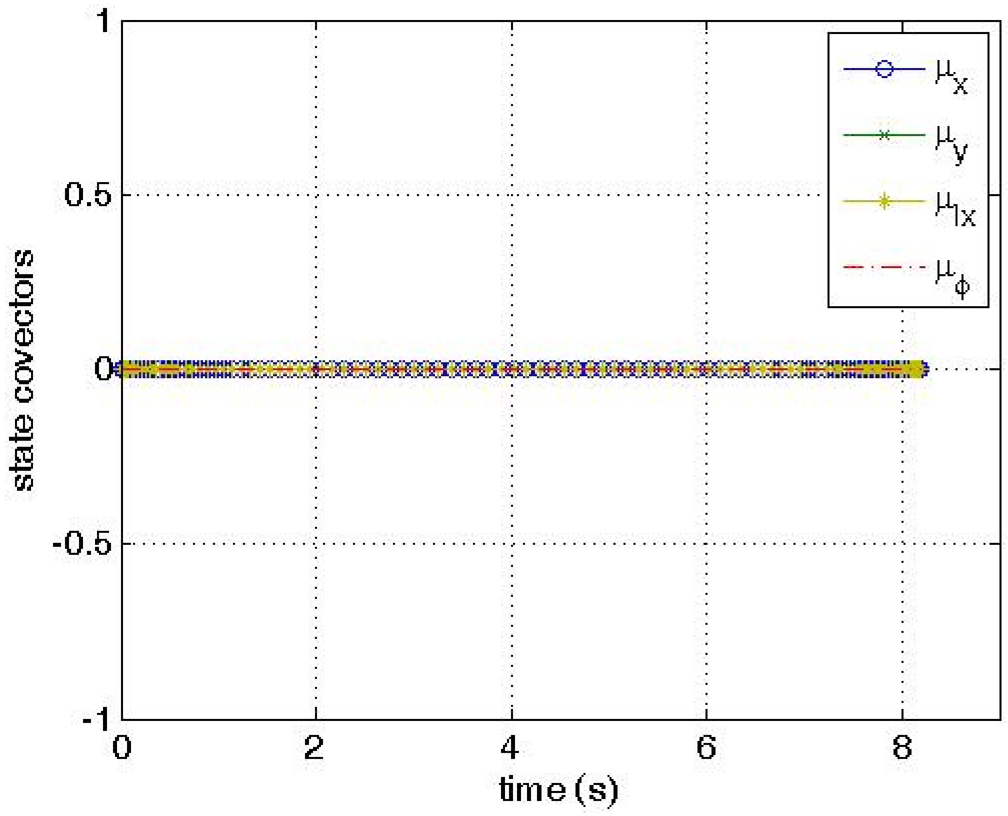


Figure 62. State Covectors.

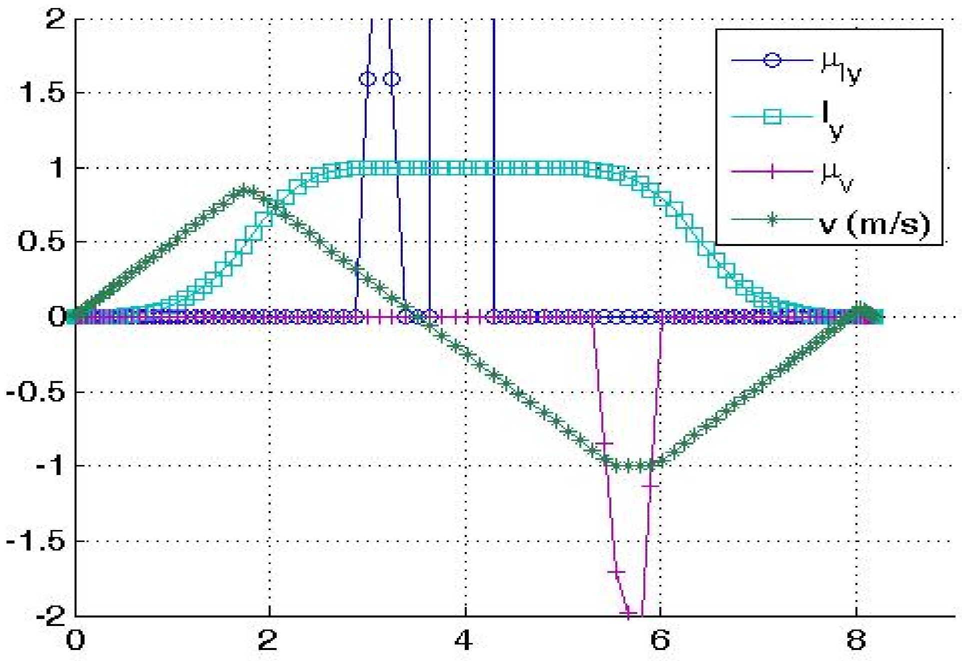


Figure 63. Relationship between μ_{I_y}, μ_v and I_y, v.

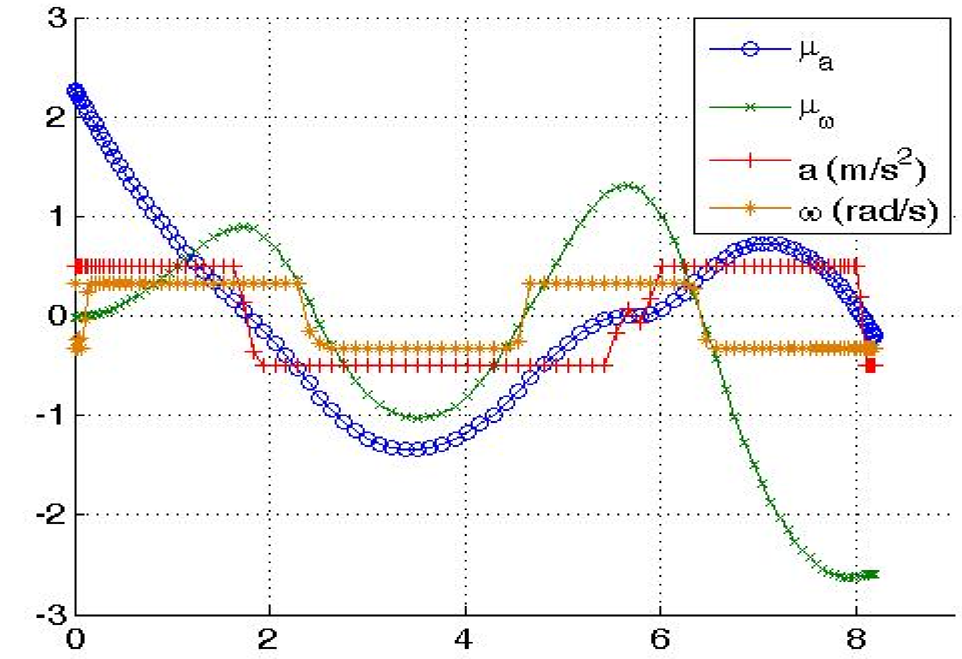


Figure 64. Relationship between μ_a, μ_ω and a, ω.

Applying the HMC again results in equation set (24). Figures 65 and 66 show the costates for this problem. Comparing the costates $\lambda_v, \lambda_\varphi$ to the control covectors μ_a, μ_ω validates equation set (24) and is shown in Figure 67.

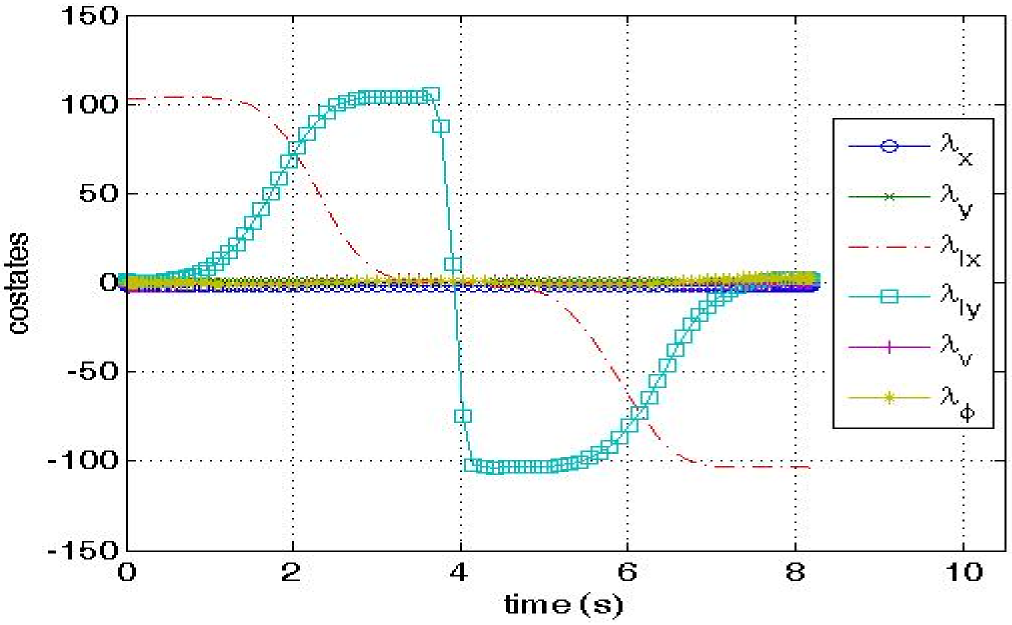


Figure 65. Costates (Small Scale).

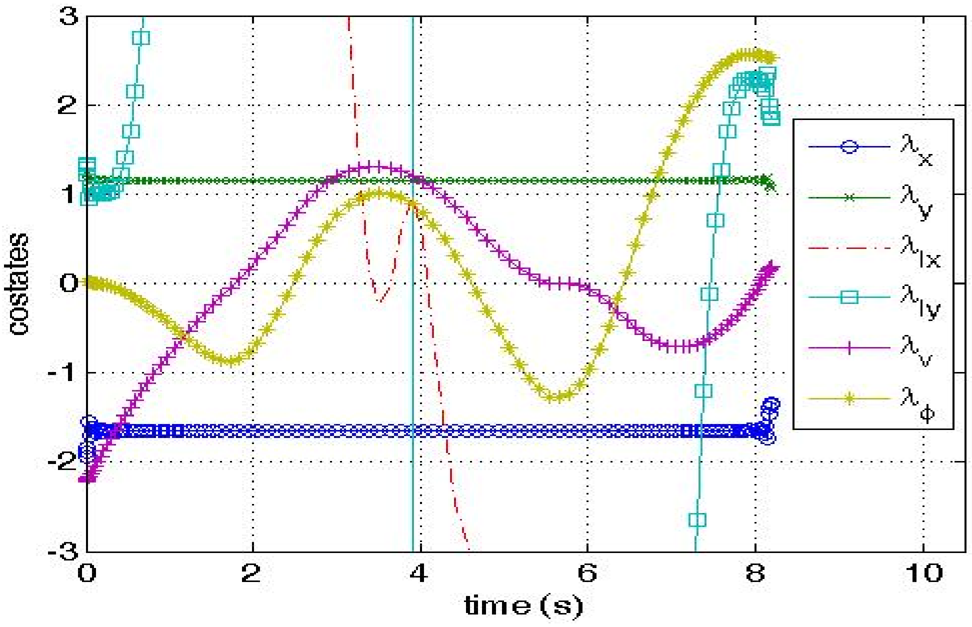


Figure 66. Costates (Large Scale).

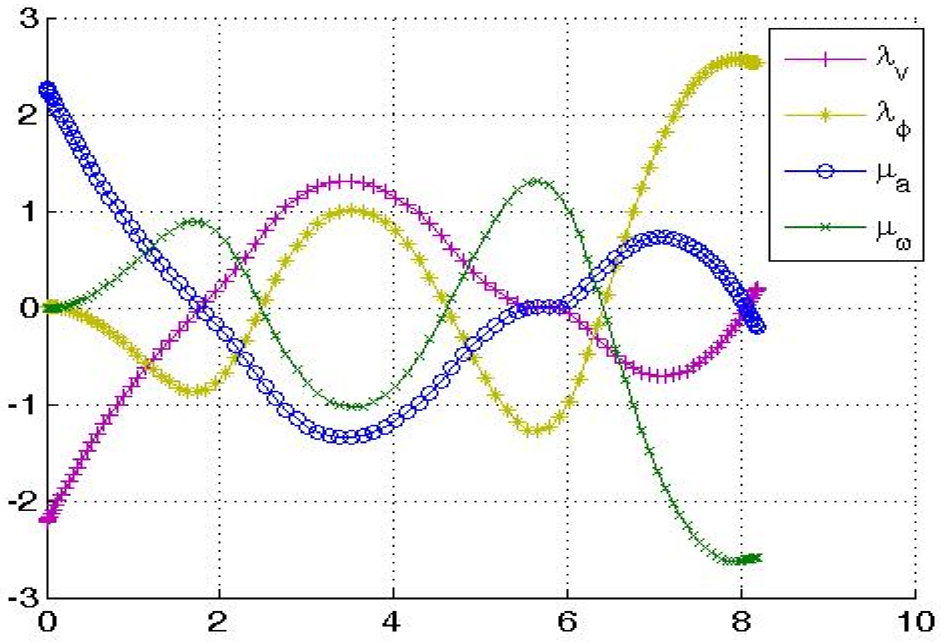


Figure 67. Comparison of $\lambda_v, \lambda_\varphi$ with μ_a, μ_ω.

Calculating the Adjoint Equation results in:

$$\begin{aligned}
\dot{\lambda}_x &= -\mu_x \\
\dot{\lambda}_y &= -\mu_y \\
\dot{\lambda}_{I_x} &= -\lambda_x v - 2\lambda_{I_y} v \tan(\varphi) - \mu_{I_x} \\
\dot{\lambda}_{I_y} &= -\lambda_y v + 2\lambda_{I_x} v \tan(\varphi) - \mu_{I_y} \\
\dot{\lambda}_v &= -\lambda_x I_x - \lambda_y I_y + 2\lambda_{I_x} I_y \tan(\varphi) - 2\lambda_{I_y} I_x \tan(\varphi) - \mu_v \\
\dot{\lambda}_\varphi &= 2v\lambda_{I_x} I_y \sec^2(\varphi) - 2v\lambda_{I_y} I_x \sec^2(\varphi) - \mu_\varphi
\end{aligned} \tag{46}$$

Since μ_x, μ_y are always zero (from Figure 62), it follows from equation set (46) that λ_x, λ_y are constants, and this can be validated using Figure 66. The next step is to evaluate the Endpoint Lagrangian ($\bar{E}$), which becomes:

$$\begin{aligned}
\bar{E} = t_f + \nu_{t_0} t_0 + \nu_{x_0}\left(x(t_0) - 5\right) + \nu_{y_0}\left(y(t_0) - 5\right) + \nu_{I_{x0}}\left(I_x(t_0) - 1\right) + \nu_{I_{y0}} I_y(t_0) + \ldots \\
\ldots + \nu_{v_0} v(t_0) + \nu_{\varphi_0}\varphi(t_0) + \nu_{x_f}\left(x(t_f) - 5\right) + \nu_{y_f}\left(y(t_f) - 4\right) + \nu_{I_{xf}}\left(I_x(t_f) - 1\right) + \ldots \\
\ldots + \nu_{I_{yf}} I_y(t_f) + \nu_{v_f} v(t_f) + \nu_{\varphi_f}\varphi(t_f)
\end{aligned} \tag{47}$$

The Endpoint Lagrangian is used to establish the Transversality Conditions, which provide nothing useful and so will not be included here. The Hamiltonian Value Condition is again given by equation (28). Combining that result with the Hamiltonian Evolution Equation results in $H = -1$ throughout the vehicle's maneuver. Figure 68 validates this conclusion.

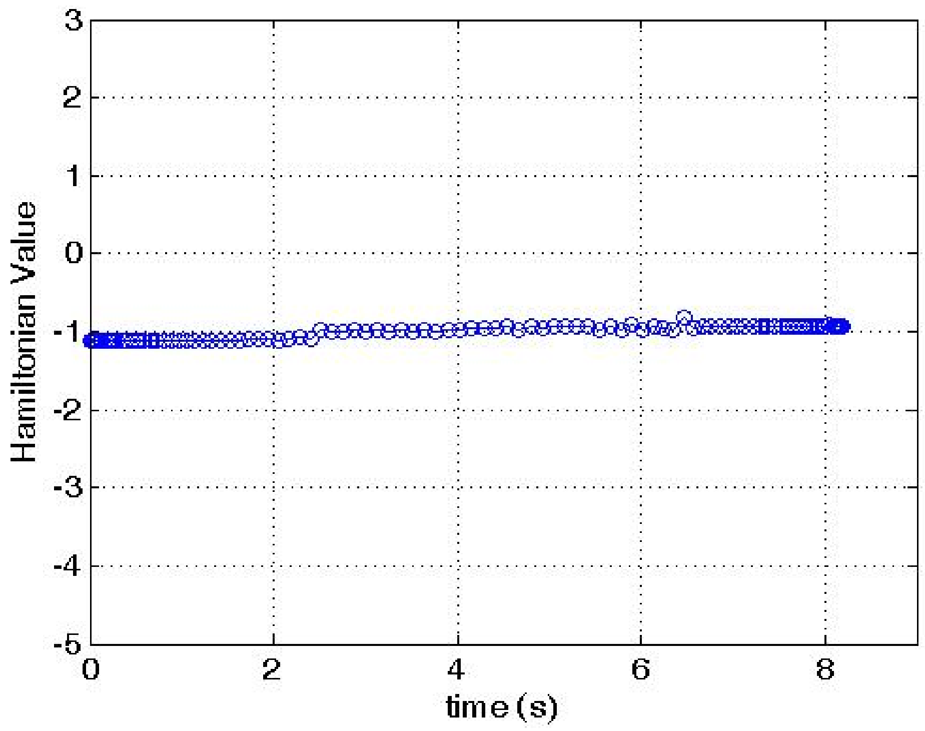


Figure 68. Hamiltonian for Alternate Problem Formulation.

The final step in determining the current extremal as the locally optimal solution is to verify Bellman's Principle. The same process was used as in the original problem formulation. Figure 69 shows the *x* vs *y* trajectory of the complete problem with the 1st half and 2nd half problems superimposed on the same graph; and it can be seen that the pieces add up to the whole. The same holds true for the remaining states I_x, I_y, v, φ as shown in Figure 70. The cost of the 1st half problem was 4.03; the cost of the 2nd half problem was 4.16; and adding the 1st and 2nd half costs comes out to 8.19, which exactly matches the total problem cost.

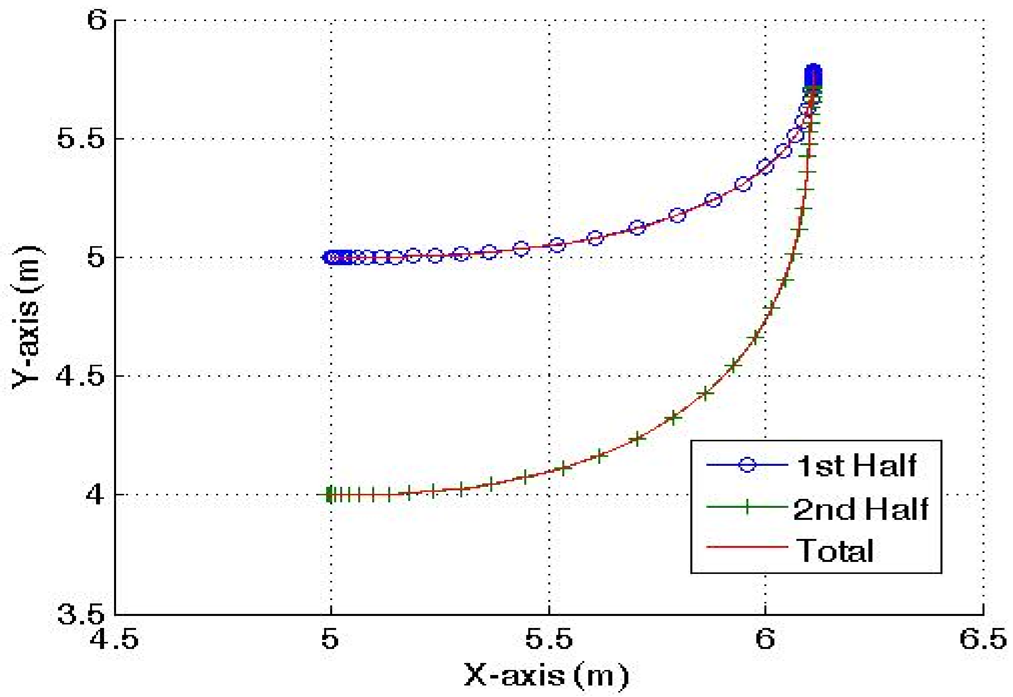


Figure 69. Bellman's Principle (*x* vs *y*).

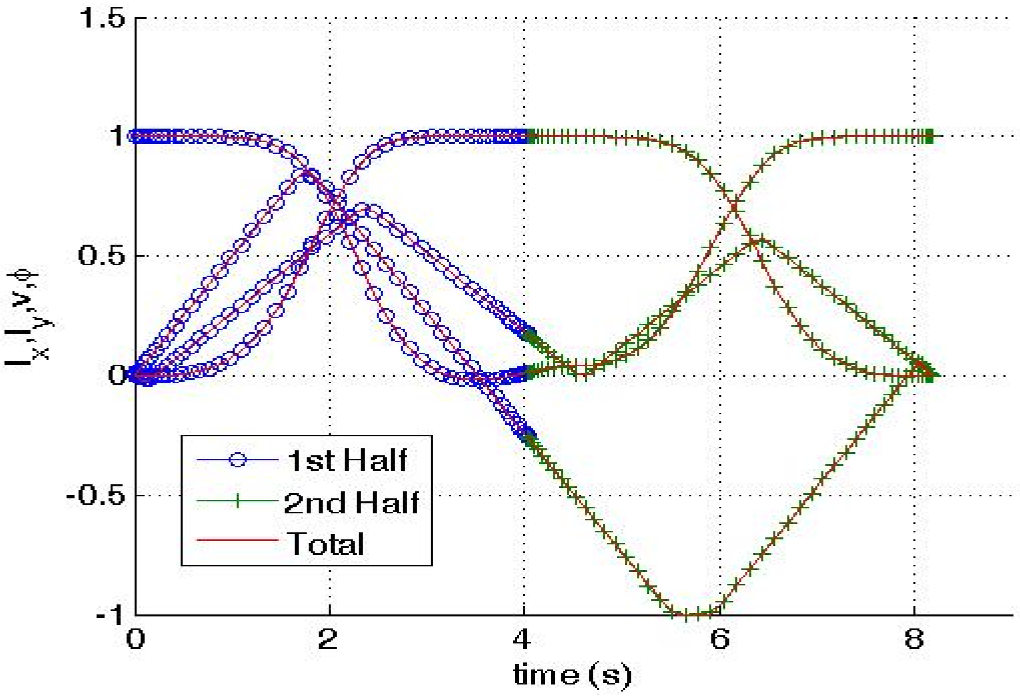


Figure 70. Bellman's Principle (I_x, I_y, v, φ).

F. ONE LAST SCENARIO

The intent here is to show one last scenario that demonstrates the options presented thus far. An obstacle-rich scenario that requires considerable maneuvering to reach the goal. Figure 71 shows the trajectory of a vehicle using the original formulation and the endpoint conditions provided by equation set (48).

$$\underline{x}_0 = \begin{bmatrix} x_0 \\ y_0 \\ \theta_0 \\ v_0 \\ \varphi_0 \end{bmatrix} = \begin{bmatrix} 12 \\ 15 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \text{and} \quad \underline{x}_f = \begin{bmatrix} x_f \\ y_f \\ \theta_f \\ v_f \\ \varphi_f \end{bmatrix} = \begin{bmatrix} 29 \\ 26 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{48}$$

There is an extra loop at the end of the vehicle's trajectory, which comes from the fact that the vehicle (in escaping the maze) has made a complete revolution. This results in effectively winding up the orientation variable (θ), which requires the turn-around loop at the end in order to unwind θ back to $\theta_f = 0$.

The trajectory of Figure 72 shows no extra loop at the end. This can be achieved a number of ways. If there is advanced knowledge as to the value of θ, then $\theta_f = 2\pi$ will result in Figure 72. If instead of using a specific value of θ we use $\sin(\theta_f) = 0$ and $\cos(\theta_f) = 1$, we can also achieve the trajectory of Figure 72. Finally, the alternate formulation provided in Chapter III.E will also provide the Figure 72 trajectory. The purpose here is not to promote one way as the best, but to show there are several ways to define the problem.

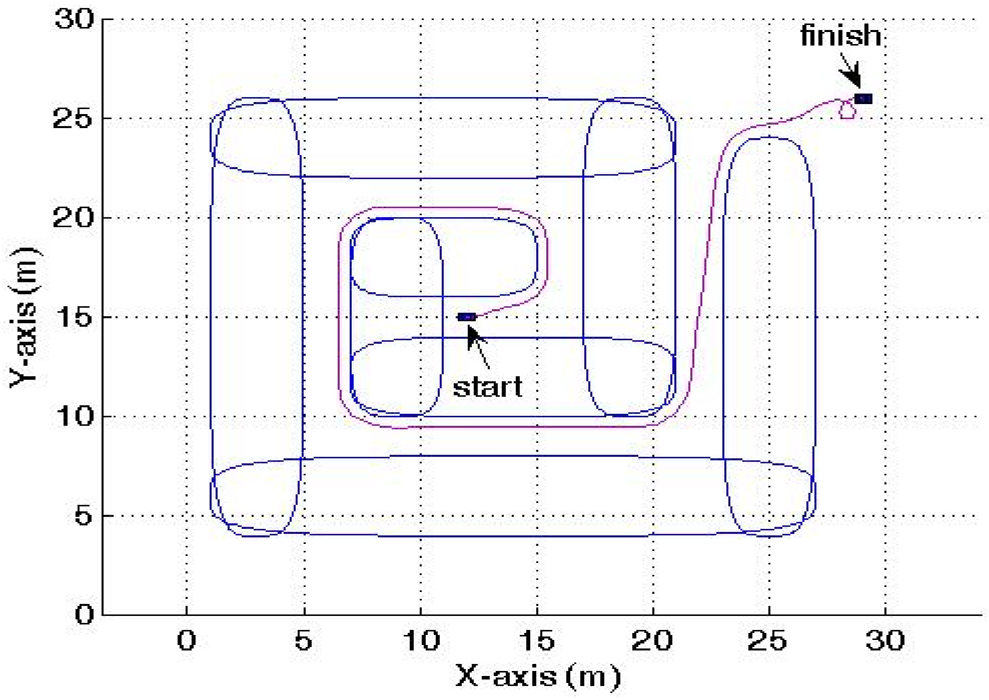


Figure 71. Final Orientation/Obstacle Scenario.

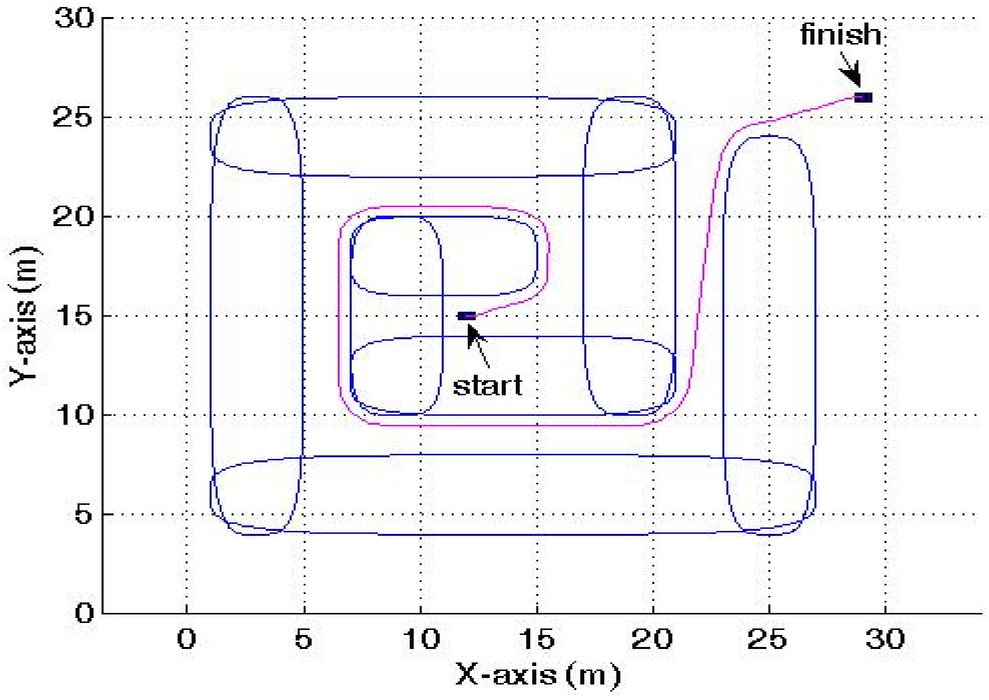


Figure 72. Final Orientation/Obstacle Scenario (Loop Removed).

THIS PAGE INTENTIONALLY LEFT BLANK

IV. FOUNDATIONS OF REAL-TIME OPTIMAL CONTROL TRAJECTORY PLANNING

A. BACKGROUND

The problems solved thus far have been strictly single-run open loop problems. They assumed an *ideal, static* world. A scenario was presented; the problem was formulated and fed into DIDO, which then calculated a locally optimal control solution. That control solution was then propagated to show its feasibility. This method works great only under certain assumptions: 1. The environment is exactly known and does not change while the vehicle carries out its trajectory. How could the vehicle react to unforeseen events if it only has one control solution that it must carry out to its end? 2. The mathematical model of the vehicle is perfect and its sensors are 100% accurate, resulting in the exact planned trajectory being followed. 3. No external forces act to alter the vehicle's trajectory or the environment it is in. These assumptions are highly unrealistic. We seek a vehicle with an algorithm that can take the initial planned trajectory as a starting point, use its sensors to continuously update its position and surrounding environment, and *react* to the changes by planning new (updated) trajectories. Without this ability, the smallest of errors would keep growing, ultimately resulting in the vehicle being so far from its originally planned trajectory that it would fail to reach its goal.

B. THE REAL-TIME CONCEPT

What is meant by real-time path planning? This has already been alluded to in the discussions of the various path planning methods. The desire is to be able to update the vehicle's position and surrounding environment and subsequently replan its trajectory while the vehicle is in motion to the goal. The speed of calculation (run time) is paramount to real-time operation. It was proposed earlier that run times under *0.5 sec* would be sufficient for a vehicle with a maximum speed of *1 m/s*. It was also argued that since a 100 fold speed up in run time is possible with certain improvements that have not yet been implemented, it would be sufficient in this work to show run times under *50* *seconds*. The initial offline run can take as long as is required to calculate the initial trajectory, since the vehicle is not yet in motion. However, once the vehicle is sent on its mission, subsequent run times must meet the speed requirement to guarantee the trajectory is followed closely and environmental changes are tracked swiftly enough.

It has already been demonstrated that using the initial (open loop) trajectory (generated offline) as the bias for another run can produce run times of sufficient speed for the scenarios covered thus far. Subsequently, each real-time solution can be used as the bias for the next real-time run, thus maintaining solution speed and autonomy. This approach involves using the optimal control algorithm in a feedback form (closed loop), thus self-generating the needed bias. Examples of this have already been presented. Also, it has already been stated that another way of lowering run times is to use the lowest number of nodes possible. It is safe to say that the effectiveness of any trajectory planning algorithm is tied to its ability to run in real time, which is heavily reliant on maintaining run times as low as possible. To achieve the lowest possible run times the algorithm must have a bias to steer each new solution, it must obtain each new solution using the lowest possible number of nodes, and it must do this autonomously.

Real-time closed loop operation allows for the algorithm to be processed at each time instant or environmental alteration, which makes it viable for use with dynamic obstacles whose future positions cannot be predicted, or pop-up obstacles not known at the start of the mission. The ability to replan the trajectory from the current vehicle position to the target at each time instant allows the vehicle to continuously update its position, thus accounting for any positional error resulting from imperfect modeling, sensor errors and the like. Continual replanning prevents any error, deviation or uncertainty from growing too large.

Closed loop operation also has the incidental benefit of correcting such open loop problems as the loop in the trajectory of Figure 71. One of the solutions to correcting for that loop was presented previously as knowing in advance what the vehicle's final orientation will be. By operating closed loop, the vehicle's current orientation is continuously fed back as the initial condition for the next iteration. The algorithm can use this knowledge to determine the correct vehicle orientation that will result in no extra loop at the end of the trajectory.

C. SOLVING A DYNAMIC PROBLEM

1. Problem Description

Figure 73a shows the initial configuration of the problem with all dimensional units in meters. This is what the vehicle knows *a priori* and is used to solve the initial planning problem. The UGV starts from *(x,y) = (0,10)* and proceeds to the target point *(x,y) = (28,10)* in minimum time, with maximum robustness, while avoiding obstacles. Obstacles 1 and 3 are stationary. At *t = 3 sec* obstacle 2 commences moving north at a rate of *1 m/s* and continues to move until *t = 7 sec*. This results in the north passage being blocked by obstacle 2 while opening up the south passage. Due to obstacle overlap, the north passage will be completely blocked at *t = 6 sec*. At *t = 25 sec*, obstacle 4 appears as a new obstacle not known *a priori*. Figure 73b shows the final configuration of the obstacle environment.

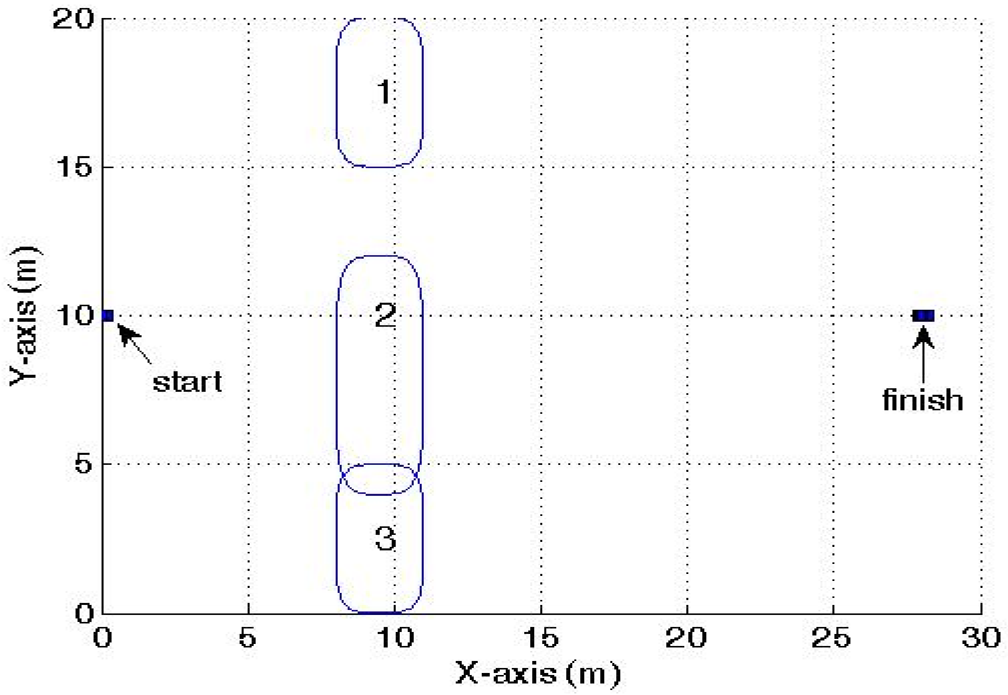


Figure 73a. Dynamic Problem (Initial Configuration).

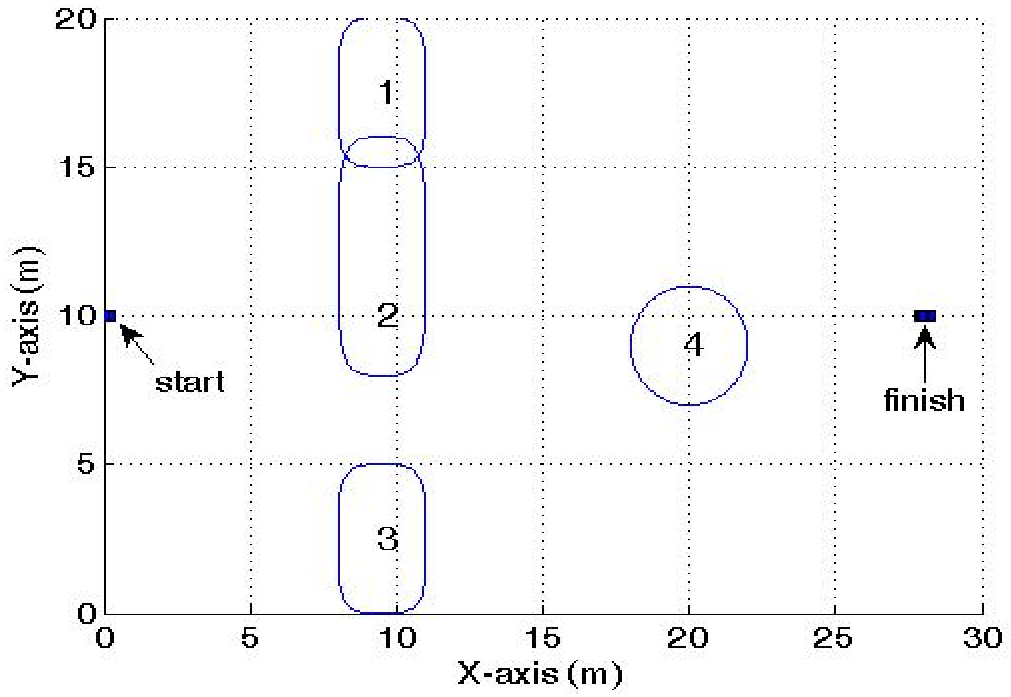


Figure 73b. Dynamic Problem (Final Configuration).

2. Problem Formulation

The original vehicle model of Chapter III.B will be used. Equation set (49) shows the path constraint equations, which were formulated by taking the obstacles of Figure 73 and expanding them by *0.5m* to account for vehicle size; that is, Equation set (49) defines the expanded obstacles of Figure 73. Note that the path constraints for obstacles 2 and 4 are time dependent, thus allowing obstacle 2 to move and obstacle 4 to appear later in the simulation as a pop up obstacle that must be avoided.

$$
\begin{aligned}
&h_1(x(t),y(t)) = \ln\left(\left(\frac{x-9.5}{2}\right)^4 + \left(\frac{y-17.5}{3}\right)^4\right) \geq 0 \\
&h_2(x(t),y(t)) = \begin{cases} \ln\left(\left(\frac{x-9.5}{2}\right)^4 + \left(\frac{y-8}{4.5}\right)^4\right) \geq 0; t < 3 \\ \ln\left(\left(\frac{x-9.5}{2}\right)^4 + \left(\frac{y-8-(t-3)}{4.5}\right)^4\right) \geq 0; 3 \leq t \leq 7 \\ \ln\left(\left(\frac{x-9.5}{2}\right)^4 + \left(\frac{y-12}{4.5}\right)^4\right) \geq 0; t > 7 \end{cases} \\
&h_3(x(t),y(t)) = \ln\left(\left(\frac{x-9.5}{2}\right)^4 + \left(\frac{y-2.5}{3}\right)^4\right) \geq 0 \\
&h_4(x(t),y(t)) = \ln\left(\left(\frac{x-20}{2.5}\right)^2 + \left(\frac{y-9}{2.5}\right)^2\right) \geq 0; t \geq 25
\end{aligned}
\tag{49}
$$

Development of the cost function requires discussion here as well. We want to minimize time as before, however we now desire to maximize robustness. By robustness we mean to say that the vehicle must be able to execute the control solution with a certain amount of uncertainty and control interpolation error, without hitting any obstacles. The uncertainty can come from numerous sources: Approximations in the modeling of the vehicle, errors in sensor data, sensor accuracy, external unforeseen forces, and the error that is inherent in executing the previous control solution while calculating an updated solution (the two will not be exactly the same). The control interpolation error comes from the fact that low node solutions are necessary to effect lower run times, which are required for real time operation. A magnified view of the trajectory of Figure 49 is shown in Figure 73c. The trajectory shows a collision with the expanded obstacle. If enough nodes are used, the trajectory can be made collision free, but at the expense of algorithm run time. The better solution to this problem is the addition of the robustness factor.

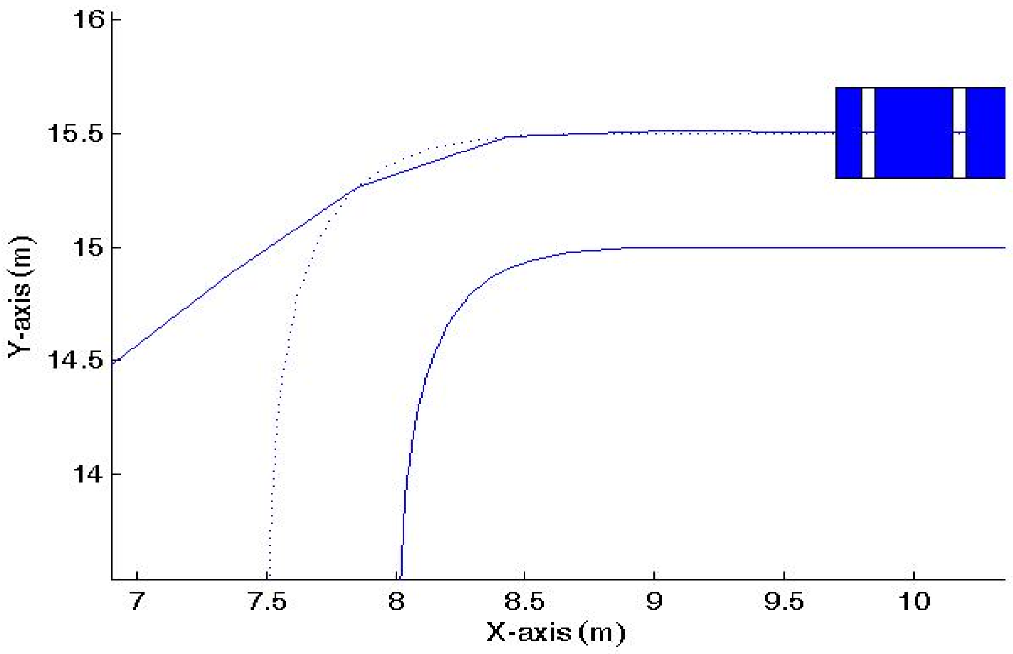


Figure 73c. Magnified View Showing Collision of the Low Node Solution.

Since we desire to minimize a cost function, it is necessary to derive a Robustness Function (*r(t)*) whose integral over time will be minimum when robustness is maximized. To derive *r(t)* we take equation (35) (which has a value of zero when on the obstacle and increases as you move away from the obstacle) and apply a double exponential operation [1]. This is done for each obstacle and the result is summed. The resulting *r(t)* is as follows with *n* equal to the number of obstacles:

$$r(t) = \sum_{i=1}^{n} \left(e^{e^{-h_i(x(t),y(t))}} - 1 \right) \tag{50}$$

In the case of a single obstacle, *r(t) = 14.15* at its center, *r(t) = 1.72* on its edge, and *r(t)* will continue to decrease exponentially to zero as the distance to the obstacle is increased. Now the cost function can be completely expressed in terms of the endpoint cost (final time) and the running cost (integral of *r(t)*) with appropriate weighting factors (w_{t_f} , w_r):

$$J[\underline{x}(\cdot),\underline{u}(\cdot),t_f] = w_{t_f} t_f + w_r \int_0^{t_f} r(t)dt \qquad (51)$$

3. The Trajectory Planning Algorithm

The vehicle's trajectory planning algorithm (Figure 74) follows the following logic:

a. The vehicle is given an instantaneous snapshot of the environment that is known prior to the start of its mission. This environment map will act as the global map that will be used to plan the trajectory to the target. It is a dynamic map that will track known environmental changes.

b. The initial path to the target (and the controls throughout that path) is solved *offline*, prior to the vehicle commencing the maneuver. This initialization calculation can take longer than future calculations in real-time, because the initialization contains only two points to aid in its solution (the initial conditions and the target conditions), and the initialization uses a higher number of nodes to establish the near optimal trajectory. This initial solution serves as the bias used to steer the first real-time (low node) calculation, which results in lowering the calculation time.

c. The following two steps are conducted simultaneously in real-time until the vehicle reaches the target point:

(1) The vehicle sensors take a snapshot of the local environment. This snapshot (along with global information received from other sources via its communications link) is used to update the global map and the current vehicle state. The new environment map and vehicle state are used to calculate the next optimal trajectory. This calculation runs much quicker, because the data from the previous solution is used as a bias to steer the next solution, and the number of nodes is kept to a minimum.

(2) The vehicle travels toward the target by executing the controls from the previous solution. This is done in simulations through the application of control trajectory interpolation (linear) and state propagation using a Runge-Kutta algorithm. It is very important to realize here that the vehicle's position using this propagation

technique will not agree exactly with the generated DIDO solution. As a matter of fact, the error between the two will be greater when interpolating a lower number of nodes. Since we desire to use the lowest number of nodes possible when operating closed loop (so as to have the fastest run times), the error between the propagated path and the DIDO solution can be significant. This 'propagation error' (as it will be called) will be used in this work, as in [1], to represent the uncertainty that a real vehicle would experience.

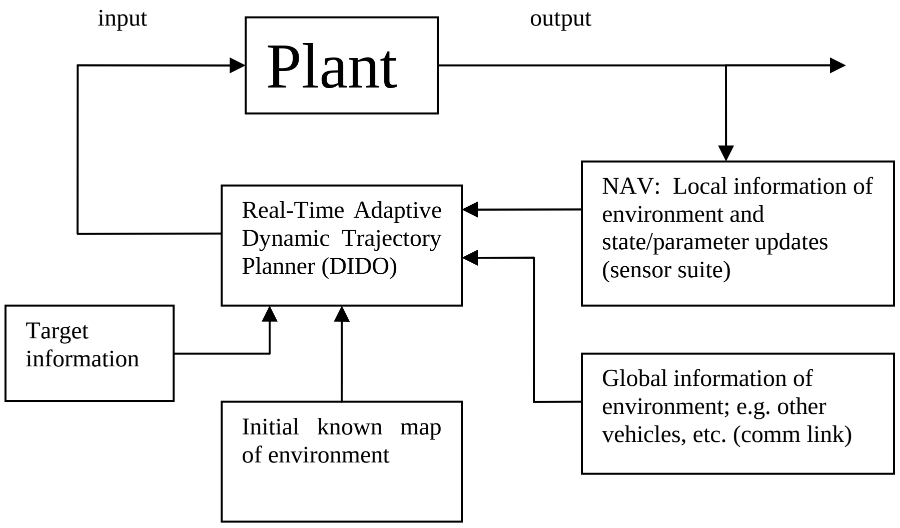

input
Plant
output
NAV: Local information of environment and state/parameter updates (sensor suite)
Real-Time Adaptive Dynamic Trajectory Planner (DIDO)
Target information
Global information of environment; e.g. other vehicles, etc. (comm link)
Initial known map of environment

Figure 74. Trajectory Planning Algorithm Logic.

4. Simulation Results and Analysis

Simulations were performed in steps to build up from the simple to the more complex scenario described in Chapter IV.C.1. Using the initial environment map of Figure 73a, the 60-node open-loop optimal trajectory is shown as a solid line in Figure 75. This solution was obtained knowing only the start and end conditions *a priori*, and using only minimum time as the cost. The robustness factor is excluded from the cost in this initialization run, because the vehicle is not yet moving, therefore there is no propagation error. The resulting solution will be the ideal time optimal solution. The use of 60 nodes, however, is not necessary in providing a good bias for subsequent real-time runs. One key aspect of pseudospectral (PS) methods that is utilized in trajectory planning is its high convergence rate. That is, in a PS method only a few discrete points, or nodes, are needed to reconstruct accurate continuous-time solutions. For the problem at hand, a mere 15-point solution is quite adequate to meet all the performance specifications. This feature is demonstrated in Figure 75 where a 15-node solution (points marked with circles) is superimposed on the 60-node solution. That the 15-node solution is almost on top of the 60-node solution indicates that the PS solution has practically converged with 15 nodes. Theorems are proven in [67, 68] showing that the interpolation based on the Legendre-Gauss-Lobatto (LGL) node points converges to the true solution of the continuous-time problem, not only at the nodes but also at anywhere between the nodes. Hence, we use a higher number of nodes (60 in this case) to establish the initial trajectory, and the 15-node solution is used as the bias to steer subsequent real-time (closed loop) calculations. The overall optimal maneuver time is *30.5 sec* and the solution satisfies all necessary conditions for optimality. The use of a low number of nodes (15 in this case) will help greatly in keeping run times to a minimum. We must keep in mind here that 15 nodes may not always work when operating in real-time. The number of nodes should be kept as low as the problem will allow, and this is determined by the population and complexity of the obstacle environment. Determining the correct number of nodes for the given situation will be discussed later.

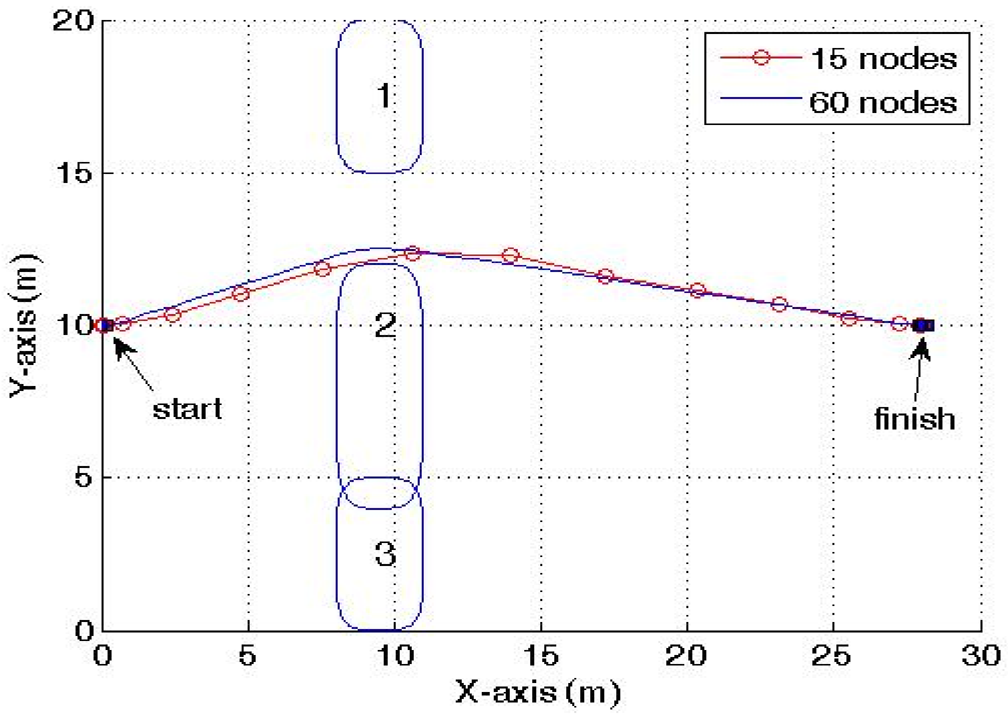


Figure 75. Initial 15 and 60 node *offline* runs.

The trajectory planning algorithm was then run in real-time (closed loop) using the starting configuration of Figure 73a. The bias used was that of the 15-node initialization run shown in Figure 75. All obstacles remained stationary and no pop up obstacles were encountered. Each new DIDO-generated solution was a 15-node solution. The robustness factor is included in the cost when operating in real-time. It was therefore necessary to determine the weightings (w_{t_f}, w_r) on the endpoint and running costs. Rather than making it necessary to fine tune both weighting factors, a weighting of $w_{t_f} = 1$ was chosen for the endpoint cost. Numerous solutions were generated until a w_r could be found that would allow the generation of feasible solutions (no collision occurred) for any run time of *0.5 sec* or less; that value was found to be $w_r \geq 1/7$. The resulting cost function is:

$$J[\underline{x}(\cdot), \underline{u}(\cdot), t_f] = t_f + \frac{1}{7}\int_0^{t_f} r(t)dt \qquad (52)$$

Since the factor of 100 speed up in calculation time has not yet been realized, run times were artificially limited. Table 3 shows the solutions that were generated for various run times. Open loop runs are not given a maximum allowed run time since there is no time limit to calculating a solution offline. Closed loop runs do not show a running or total cost, because those parameters apply only to single DIDO calls, and closed loop solutions are a compilation of many DIDO calls as the algorithm is run over and over in feedback form. From Table 3 it can be seen that actual run times are sufficiently low (*<50 sec*) to allow real-time operation. Once the weighting of the running cost was determined, an open loop 60-node run was conducted to see how much effect the running cost had. We see that the running cost is small when compared to the endpoint cost (t_f), and it only raises the maneuver time by *0.3 sec*. Figure 76 shows the vehicle trajectories corresponding to each closed loop solution presented in Table 3. The trajectory with the *0.4 sec* run time is chosen here as the best compromise between the lower run times and the higher (*0.5 sec*) run time. This backs up the proposal that run times under *50 sec* (which becomes under *0.5 sec* with the factor of 100 improvement in algorithm speed) will be sufficient for real-time operation. Figure 77 shows a comparison of the closed loop trajectory (using *0.4 sec* run times) with the open loop *offline* solutions without a running cost (initialization run, Figure 75) and with a running cost. The closed loop trajectory matches very closely to the open loop trajectory with a running cost; another reason why the trajectory with the *0.4 sec* run time is the most suited. The difference in maneuver time (t_f) between the open loop runs is attributed to the running cost. The problem ceases to be purely a minimum time problem with the addition of the robustness factor (running cost). The vehicle takes a wider turn around obstacle 2 in order to optimize the total cost. The maneuver time rises slightly in order to simultaneously minimize the running cost (maximize robustness). The end result is to provide the minimum total cost.

Table 3. Trajectory Comparison (Static Problem).

Open/Closed Loop (# of nodes)	**Max Allowed Run Times (sec)**	**Actual Run Times (sec)**	t_f	$\frac{1}{7}\int_0^{t_f} r(t)dt$	**Total Cost**
Open (60)	n/a	43.7	30.5	n/a	30.5
Open (60)	n/a	27.3	30.8	0.4	31.2
Closed (15)	0.1	0.31 – 1.80	30.6	n/a	n/a
Closed (15)	0.2	0.19 – 1.66	30.6	n/a	n/a
Closed (15)	0.3	0.20 – 1.92	30.6	n/a	n/a
Closed (15)	0.4	0.48 – 1.61	31.0	n/a	n/a
Closed (15)	0.5	0.47 – 1.70	32.0	n/a	n/a

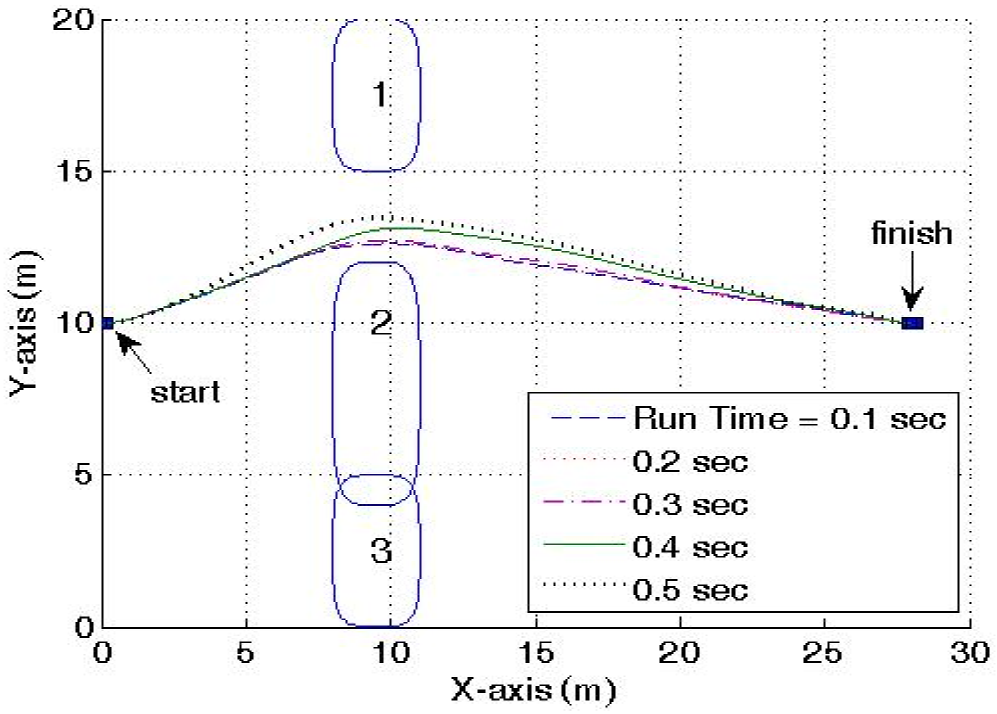


Figure 76. Closed Loop Trajectories from Table 3.

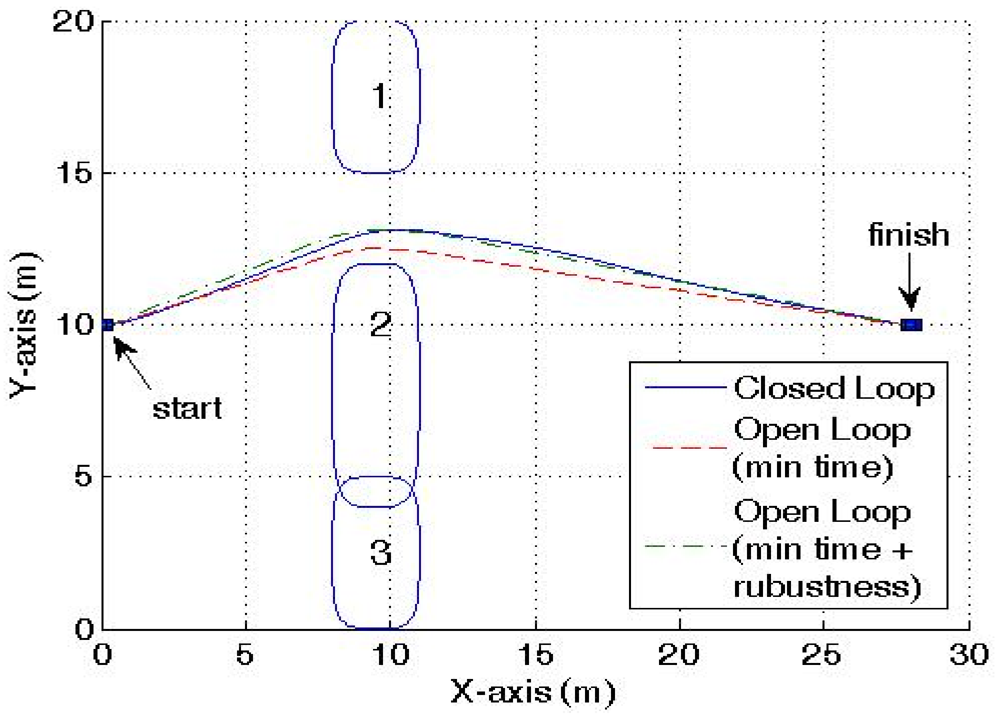


Figure 77. Closed Loop vs. Open Loop Trajectory Comparison.

Operating in real-time requires the addition of a step in the trajectory planning algorithm which checks for the dynamic feasibility of each newly generated solution. As was described in the previous chapter, showing the feasibility of the generated control solution can be done by control trajectory interpolation and state propagation using a Runge-Kutta algorithm. If the initial conditions and system dynamics can be propagated using the optimal control solution, and it matches the DIDO generated trajectories, then the control solution is deemed feasible. The feasibility check is conducted at each iteration by interpolating (spline) the control solution and propagating the vehicle's path. The deviation of each propagated point from the DIDO generated solution is added up in order to establish the norm of the deviation. If the norm of the deviation exceeds a maximum value, the trajectory is not feasible. If the trajectory is found to be infeasible, the vehicle must stop and repositions to where it was when it started that iteration of the algorithm. It performs this repositioning because it was the last position the vehicle had a feasible solution. The details of this repositioning are revealed later in this section. This check for infeasibility improves the autonomy of the trajectory planning algorithm.

The results provided thus far in simulations brings the following question to light regarding the use of the algorithm on a real UGV: What if the algorithm takes longer than the maximum allowed run time to produce a solution? It has already been shown in simulations that propagation error can degrade the vehicle's trajectory an undesirable amount if the run time is too long. It follows that if the run time exceeds the maximum allowed, the uncertainties that produce positional error in a real vehicle will result in undesirable trajectories. The solution to this question is quite simple; if the vehicle has not received an updated control trajectory within the prescribed maximum allowed algorithm run time, the vehicle stops at its current position and waits for the updated solution. For this reason, it is important that the algorithm can operate with sufficient speed, or the vehicle will continually stop and go as it waits for updated solutions.

In the next scenario, obstacle 2 moves north during the vehicle's motion and blocks the north passage. The cost function of Equation (52) was utilized and run times were artificially limited to *0.4 sec*. The weighting on the running cost and the maximum allowed run time of *0.4 sec* were determined in the previous scenario when the obstacles were not in motion. In the current scenario, the environment is dynamic and requires the vehicle make a steeper approach on obstacle 2, resulting in a sharper turn. This maneuver could not be accomplished with the low weighting of the running cost; that is, the weighting of 1/7 in Equation (52) does not provide sufficient robustness to prevent collision. A more detailed study will be conducted later in order to establish proper weightings on the running cost so as to make it less scenario specific. Under the current scenario, numerous solutions were generated until a w_r could be found that would allow the generation of feasible solutions (i.e., no collision) for any run time of *0.4 sec* or less, which will allow run times to be of any value up to the maximum allowed of *0.4 sec*; that value was found to be $w_r \geq 1/4$. The resulting cost function is:

$$J[\underline{x}(\cdot), \underline{u}(\cdot), t_f] = t_f + \frac{1}{4}\int_0^{t_f} r(t)dt \qquad (53)$$

The results from the trajectory planning algorithm using the cost function of Equation (53) are shown in Figure 78. It is important to note that no prediction is involved in determining the future position of obstacle 2 and the vehicle changes its course autonomously. Each new run uses the snapshot of the environment taken just prior to that run. So, during the time it takes to complete the new solution, although the obstacle is moving continuously, the vehicle maneuvers based on the previous solution with no knowledge of obstacle motion until the next snapshot is taken and the vehicle sees the obstacle has changed position. This can be seen by noting the concavity of the vehicle trajectory in Figure 78 during the first seven seconds. It shows the vehicle trying to head further north to follow over the top of obstacle 2 as it moves north. One might be tempted to predict the position of the obstacle, which in this scenario would have resulted in a much earlier turn to the south by the vehicle. This kind of prediction can come from knowledge of the obstacle's course and speed. The effect of having more information on trajectory planning will be investigated later in this work.

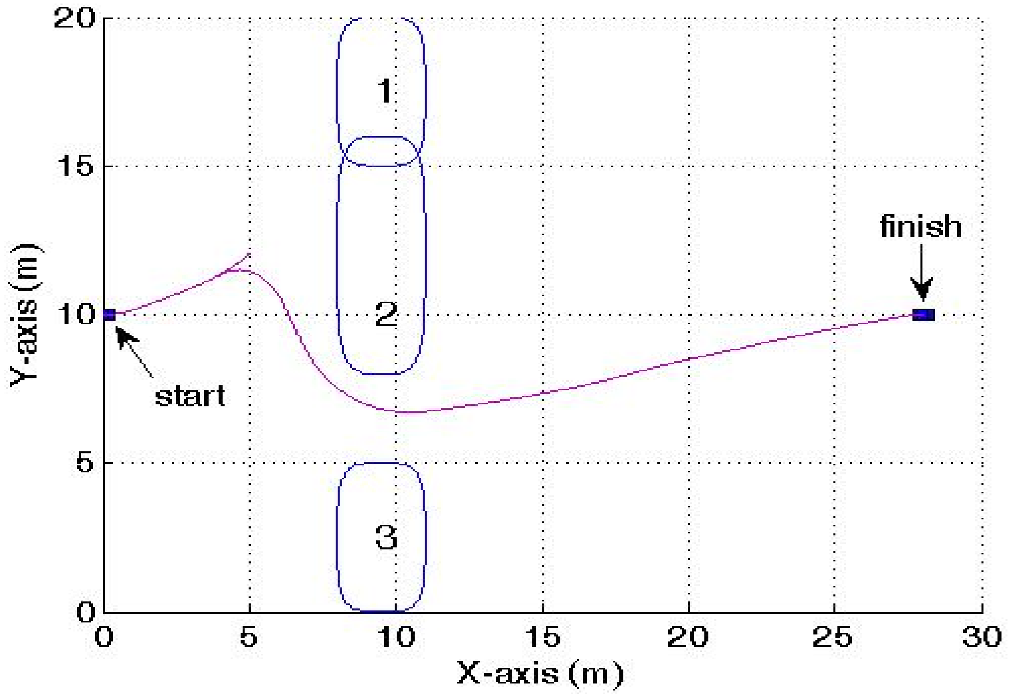


Figure 78. Closed Loop Trajectory with Obstacle 2 Movement.

The results obtained in Figure 78 required the addition of a new decision path in the trajectory planning algorithm that prevented the use of a northerly bias (that of the previous run) once the passage was blocked by obstacle 2. DIDO (as with all numerical methods) is very sensitive to the bias it is given; that is, if the bias no longer provides a viable solution, neither will DIDO. By viable, we mean to say the trajectory is collision free and ultimately reaches the target (i.e., safe passage exists). It becomes necessary to add a collision check to each iteration of the algorithm. The collision check occurs just after each DIDO call. The purpose of the collision check is to verify the DIDO generated solution does not produce a trajectory that is not viable, because that non-viable solution would become the bias for the next iteration. The new decision path flows as follows: 1. A DIDO call is made during an iteration of the algorithm. 2. Instead of checking the DIDO generated points for collision, we interpolate the control solution and propagate the vehicle's path using a high number of points. These points are then checked for collision with obstacles. The issue here is that in real-time DIDO uses low node solutions to solve for the trajectory. Remember that DIDO is a discrete numerical method. It could conceivably find a 15 point solution with obstacles that come up in between its points, and thus show no collision. Figure 79 shows the propagated path superimposed on the DIDO generated trajectory once the north passage becomes blocked. The 15 nodes of the DIDO solution do not collide with any obstacles. The propagated path (using more points) results in collision and in the vehicle determining that the current bias is no longer safe. The collision analysis is performed using the expanded obstacles in order to account for vehicle size. 3. Once the vehicle senses its current trajectory (and bias) is no longer safe, it stops (if it has not already stopped due to calculation time exceeding maximum allowed run time) and repositions to where it was when it started that iteration of the algorithm. This repositioning is executed because while the algorithm was calculating the trajectory and determining it was infeasible or unsafe, the vehicle was traveling along the previous trajectory which is what the algorithm uses as the bias. If the bias is found to be no good, then the vehicle should not have been traversing it, and thus it backtracks to its previous position. 4. From this position the vehicle will wait as the algorithm calculates a new bias. This calculation can be done using only the current position of the vehicle and its target position as its starting bias for the DIDO call. Although sensitive to being steered by a given bias (as with all numerical methods), DIDO is also less dependent than other numerical methods on the need for a bias; that is, it can still solve for the trajectory with less bias input (in this case the start and goal locations vice a 15 node previous run). With less bias points to steer the solution, it will take a higher node solution (60 nodes were used in this scenario) and a longer run time to determine the new trajectory (and thus the new bias for the next run). However, since the vehicle is not moving, it can wait for the solution to be generated. The new bias (DIDO solution and propagated path) is shown in Figure 80. The DIDO solution in Figure 80 shows a viable solution. The propagated path confirms the solution is feasible and can be used as a bias for continued real-time operation. 5. Closed loop operation will then continue again using 15 node runs. Why does the new southerly trajectory still require the vehicle to head north prior to turning south? The vehicle constraints incorporated into the optimal control problem formulation prevent it from simply making a sharp turn to the south. The front wheels can not exceed their maximum turn rate, and the vehicle has a turning radius that must be obeyed. Figure 80 shows the path that must be taken given the vehicle constraints and the current northerly heading.

The overshoot in the trajectory that occurs around $x = 5m$ in Figure 78 exists for two reasons. First, the vehicle continues to move forward (for at least the maximum allowed run time) using the last solution as it is computing the new solution and determining its safety through collision checks. Second, once the vehicle determines the new solution is not safe, it must slow down (it cannot stop instantaneously) and then backtrack to its position prior to that time step. The overall maneuver time for the scenario shown in Figure 78 is *39.5 sec*. This maneuver time includes the time needed for the vehicle to slow down and reposition itself for the new southerly solution, and does not include the time the vehicle is stopped while it waits for the new trajectory. During closed loop operation, all actual computer run times were sufficiently low (*<50 sec*) to allow real-time operation. The addition of collision checks greatly improves the autonomy of the trajectory planning algorithm.

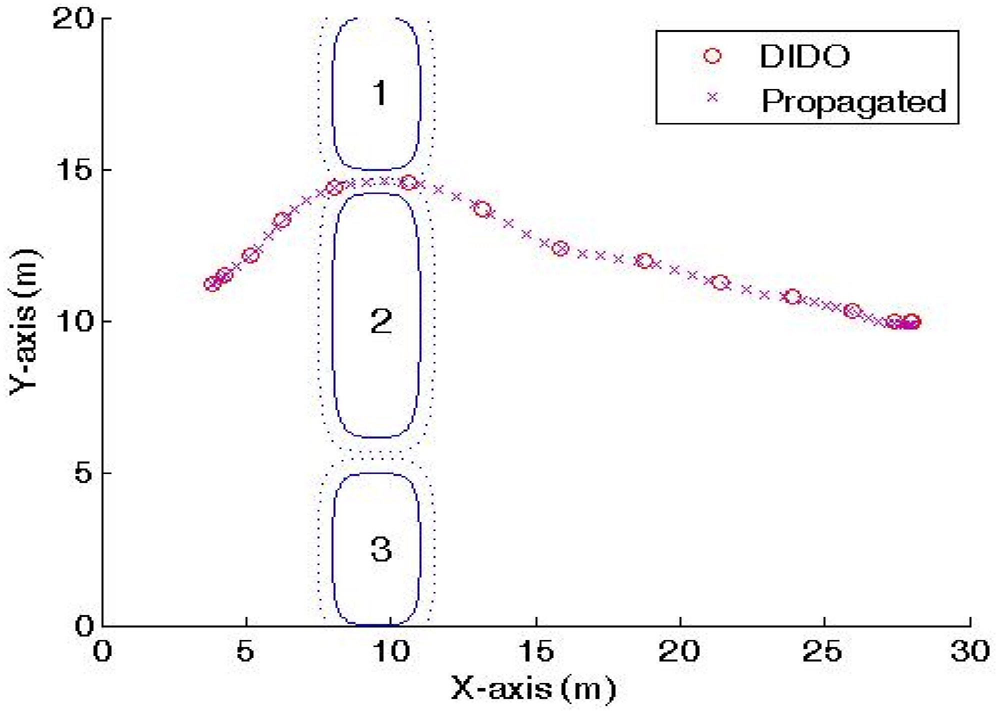


Figure 79. Algorithm Detects Collision on North Passage.

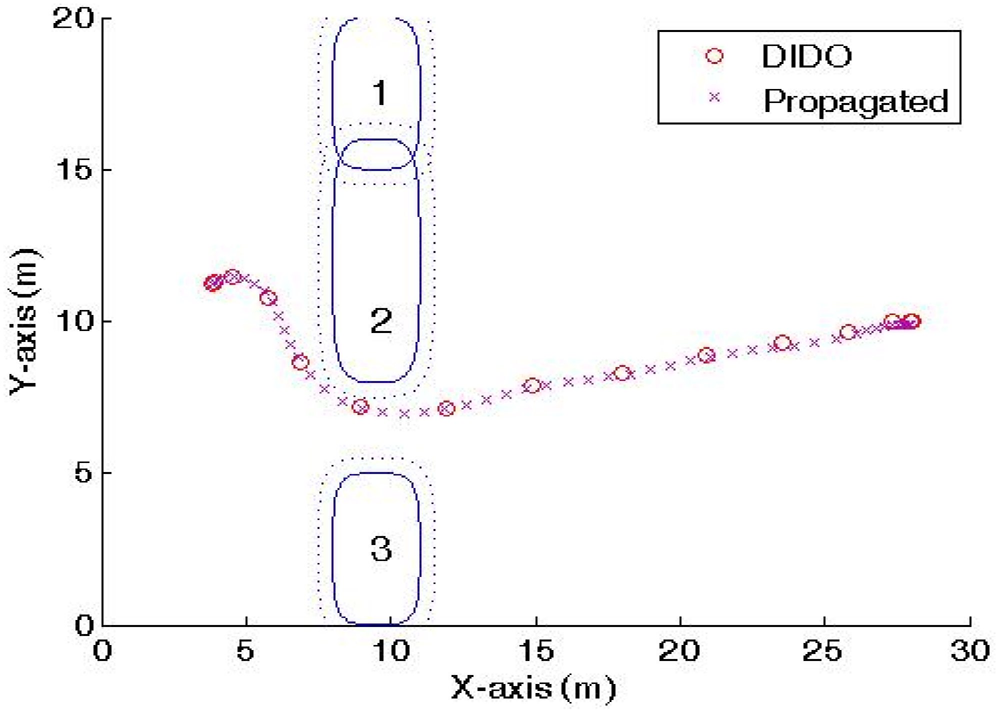


Figure 80. Algorithm Detects Clear Path to the South.

To complete the analysis of the dynamic problem in this section, the previous scenario was repeated with the addition of a pop-up obstacle at *t* = *25 sec*. The results of using the real-time optimal control technique are shown in Figure 81. Equation (53) was again used as the cost function and run times were again limited to *0.4 sec*. The overall maneuver time in this case was increased to *40.9 sec*, and all actual computer run times were sufficiently low (<*50 sec*) to allow real-time operation. Initially the vehicle heads toward the opening on the north end of obstacle 2. As obstacle 2 closes the north passage, the vehicle tries to follow it over the top until the passage is completely blocked off, at which point the vehicle autonomously finds a new optimal path to the south. Once the vehicle has passed obstacle 2 it heads straight at its goal, because it does not yet know of the existence of obstacle 4. Once its sensors pick up obstacle 4, the vehicle computes a new path around that obstacle without any operator interaction or involvement. Figure 82 shows what the path would look like if obstacle 4 had appeared sooner (*t* = *15 sec*). This new scenario has an overall maneuver time of *42.0 sec*. Combining Figures 78, 81, 82 into Figure 83 shows that the vehicle switches to the path to avoid obstacle 4 once the existence of obstacle 4 is determined. Note that once the vehicle comes around obstacle 2, the optimal path to the goal depends on how soon the planner has knowledge of obstacle 4.

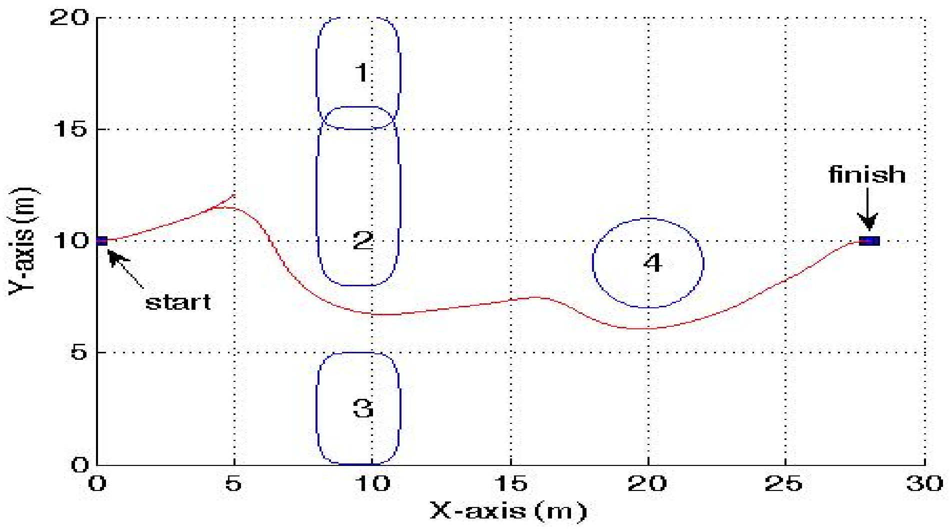


Figure 81. Vehicle Trajectory (Obstacle 4 Appears at *t* = *25 sec*).

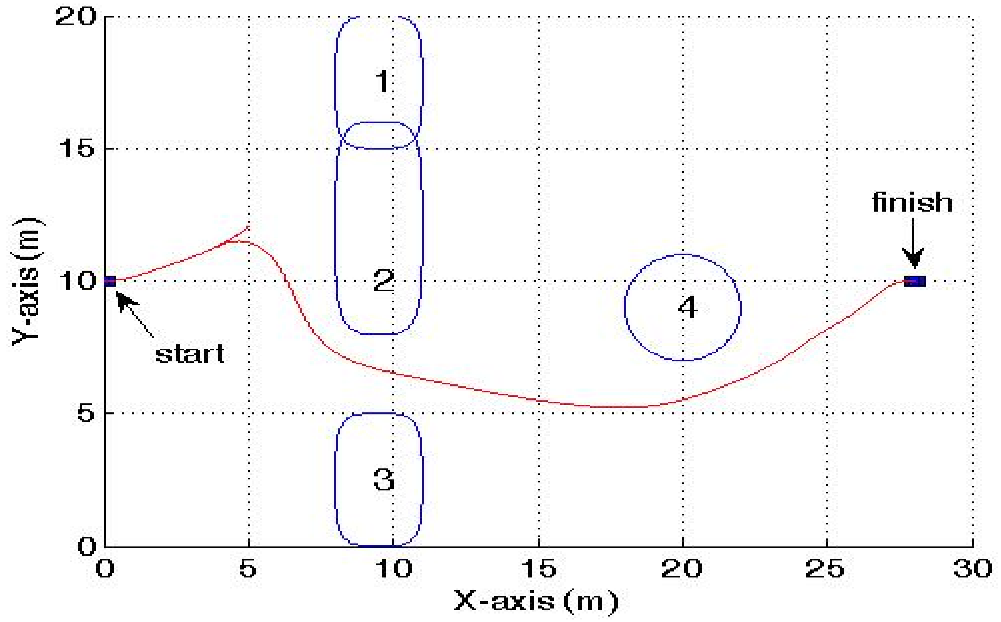


Figure 82. Vehicle Trajectory (Obstacle 4 Appears at *t = 15 sec*).

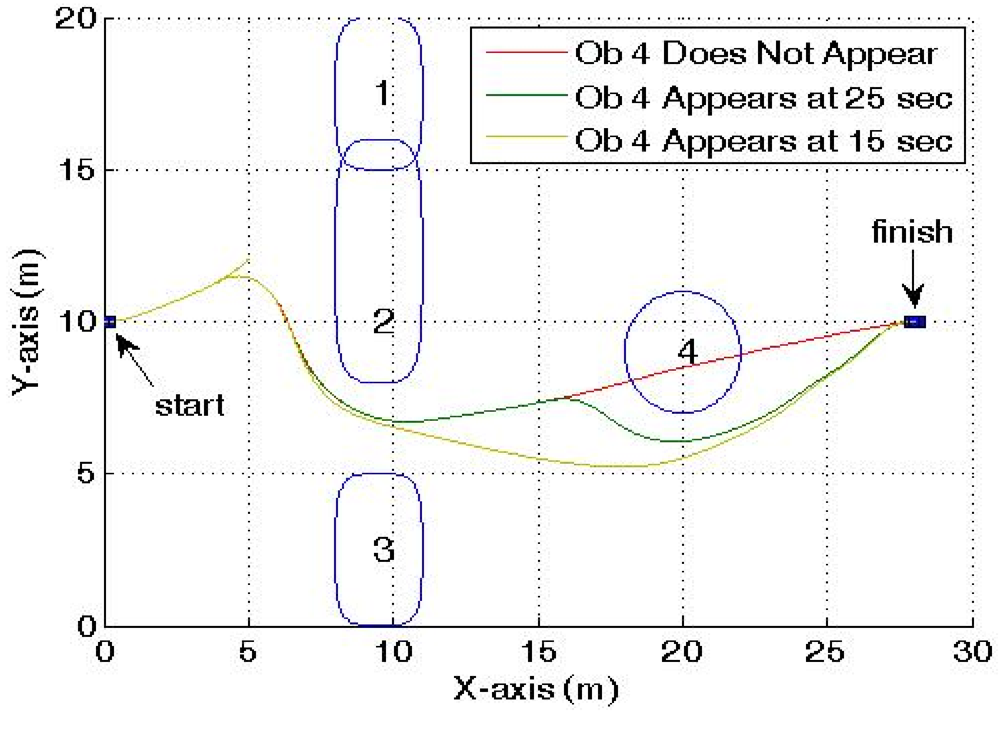


Figure 83. Vehicle Trajectories (Obstacle 4 Appearance Varies).

D. ALGORITHM ISSUES AND IMPROVEMENTS

During the analysis and solution of the example dynamic problem of the previous section, numerous issues were discovered whose solution will lead to vast improvements in the trajectory planning algorithm's robustness to solving any scenarios and its autonomy in making mid-course adjustments when the need arises. Some issues have already been addressed and solved, and some improvements have already been made in the process of solving the aforementioned dynamic problem. For example, the need to make mid-course corrections arose when the current solution (and bias for the next solution) turned out to be either infeasible or not safe. The distinction between feasibility and safety is simple. Dynamic feasibility indicates the propagated path matches the DIDO generated path; it tells us the control solution can be implemented to produce the trajectory that DIDO has solved. Safety simply indicates that the proposed trajectory is free and clear; in other words, the goal is reachable using the proposed trajectory. This resulted in modifying the algorithm to include dynamic feasibility checks and collision checks (to verify safety). An infeasible or unsafe solution requires the vehicle to stop and reposition to where it last had a feasible/safe solution.

Some other modifications include the need to stop the vehicle if run times exceed a maximum allowed limit (*0.4 sec*) (so as to prevent the propagation error from being magnified too greatly). The remainder of this chapter concerns itself with further improvements to the trajectory planning algorithm. It starts with two issues that have already been alluded to as needing further study: 1. How to determine the correct number of nodes to use per iteration in real-time. 2. How to determine the correct weighting on the running cost so as to make the tuning of the weighting autonomous to any scenario. These two issues (and more) are studied here and the solutions are automated into the trajectory planning algorithm.

1. Selection of the Number of Nodes

The most important functions in the trajectory planning algorithm are those relating to self-generating the bias and determining the lowest possible number of nodes for feasible and safe trajectories. This is because the bias and the number of nodes are the main drivers of algorithm run time. The trajectory planning algorithm has been designed to operate in a feedback mode, thus self-generating the needed bias. Up to this point, the number of nodes being used for real-time operation was manually input by the user. For improved autonomy, the required number of nodes must also be self-generated by the algorithm.

During the course of solving the dynamic problem of the previous section, it was stated that the number of nodes needed per iteration in real time is a function of the total distance the vehicle will travel and the obstacle size. The total distance the vehicle will travel is a function of the total size of the environment and the size and distribution of obstacles that the vehicle must avoid. The first step to solving this issue was to determine what the relationship is between the maximum nodal spread (maximum distance between nodes) and the size of obstacles that block the vehicle's path. Simulation results have shown that the maximum nodal spread that can be allowed is that which results in the nodal spread to obstacle size ratio of Equation (54).

$$\frac{nodalspread}{obstaclesize} \leq 0.61 \tag{54}$$

Recall that the size of an obstacle is determined by the *a* and *b* dimensions of Equation (35). The obstacle size used for Equation (54) corresponds to the minimum obstacle width in the environment; that is, twice the lowest value of all the *a* and *b* dimensions in the problem. To put it another way, the maximum distance between nodes cannot exceed 0.61 times the obstacle width. So how is the determination of the needed number of nodes automated into the trajectory planning algorithm? Prior to each iteration, the algorithm calculates two parameters: 1. The total distance the vehicle must travel to reach its goal. 2. The minimum obstacle dimension in the environment. Using the obstacle size, the nodal spread can be determined using Equation (54). Dividing the nodal spread into the total distance the vehicle will travel results in the minimum required number of nodes. This minimum number is further refined by raising it to the nearest multiple of five, which gives a buffer (margin of error) to the minimum number of nodes. Furthermore, in order to preserve nodal resolution, the absolute minimum number of nodes is set at fifteen nodes. Figure 84 shows a simple example comparing two trajectories; one using a nodal spread that obeys Equation (54) and one whose nodal spread is too large. It can be seen that if the nodal spread is too large it can result in an infeasible and/or unsafe solution. The solution is feasible when considering that DIDO only checks its discrete points (marked by circles), but when the solution is propagated, the trajectory in this case is found to be infeasible and unsafe.

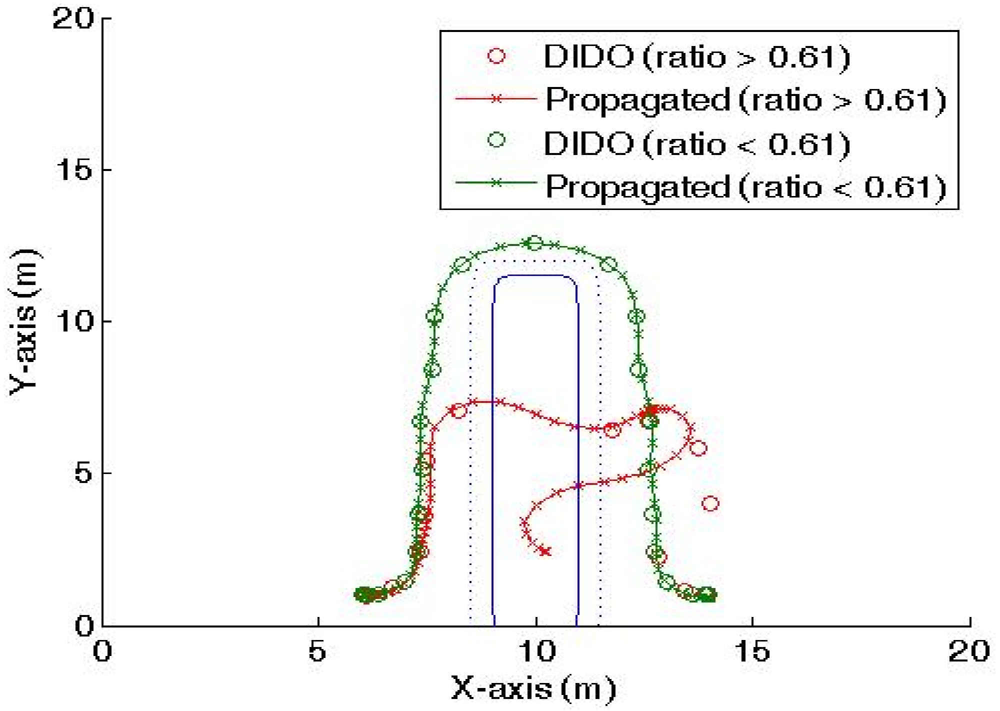


Figure 84. Trajectory Comparison using Two Different Nodal Spreads.

This capability to autonomously calculate the minimum number of nodes required at each iteration can be exploited by the user to achieve better run times (and thus better performance) if needed. The user can affect the nodal calculation by judiciously designing the obstacle environment in such a way as to raise the minimum obstacle dimension. After all, it has already been stated that exact obstacle modeling is not necessary. The failed (red) path of Figure 84 would require more nodes (and thus higher run times) to meet the Equation (54) ratio and thus give a feasible and safe path. However, the correct ratio and feasible/safe path could also be achieved by simply making the modeled obstacle wider. The one drawback to this is its effect on optimality.

The goal of the trajectory planning algorithm is not to find the time optimal path to the goal, but to find a feasible and safe path that is nearly time optimal.

2. Weighting of the Running Cost

The purpose here is to build autonomy into the trajectory planning algorithm by allowing the algorithm itself to fine tune the weighting of the running cost. So, why not just pick a large weighting that we know will provide more than enough robustness to clear all the obstacles? We must not forget that we want to maximize robustness while at the same time still minimizing time, and thus, still providing a nearly time-optimal trajectory. Figure 77 and Table 3 show the effect the running cost has on the maneuver time and the robustness. Adding too much robustness unnecessarily raises the maneuver time and causes the trajectory to be further from time optimal. Why not go the other way and use a small weighting? This would maintain a near optimal maneuver time. We can prevent the vehicle from hitting the corners of obstacles by simply raising the nodal density. This solution is also flawed, because it competes against the need for low node solutions to keep the run times down. The need for low node solutions will always exist. In the scenarios shown in this work up to this point, 15 node solutions were sufficient as the low node solutions. Hypothetically, if we desired to maintain w_r small, let us say 100 node solutions resulted in safe trajectories. Let us further say that in a few years computational advances make 100 node solutions fast enough for real time operations. This is great and would solve the simple scenarios of this work. Now expand the *30m X 20m* area used in the dynamic scenario of the previous section to an area covering *30km X 20km*; such a large area is realistic when talking about a vehicle conducting a reconnaissance mission in a large city. In order to obtain feasible and safe solutions, it could take 1000 node solutions to provide the resolution needed to navigate around obstacles when w_r is kept small. The bottom line is that no matter how fast the computational capability can be, there will always be a need to minimize the number of nodes, which results in the need to raise the robustness. We want enough robustness (large enough w_r) to effect a safe path using low node solutions, but at the same time we want to use the lowest w_r possible in order to maintain a near time optimal solution.

In the example dynamic problem of the previous section, the minimum weighting of the running cost (w_r) was *1/7* for the static case, and then *1/4* for the dynamic case. Further simulations have shown that the required value of w_r that results in a feasible and safe path varies depending on the nodal spacing, the complexity of the vehicle trajectory, the geometry of the obstacle environment and the number of obstacles. This makes the choice of w_r very difficult. It was then decided to start w_r at a low value, such as $w_r = 0.15$ (corresponding approximately to the *1/7* value determined previously), and increment it as the vehicle is traveling its trajectory until a safe path can be achieved. This required altering the collision check portion of the algorithm to distinguish between a solid collision (resulting in the determination of an unsafe path) and a grazing collision (resulting in incrementing w_r for the next run). This is done by shrinking the expanded obstacles by *35%* of their minimum dimension prior to conducting the collision checks. If a collision is indicated, it is deemed a solid collision and the trajectory is deemed unsafe. If no collision exists with the shrunken obstacles, but collision exists with the expanded obstacles, then the collision is deemed a grazing collision and w_r is incremented. Figure 85 shows an example of a grazing collision. The inner obstacle is the shrunken obstacle while the outer is the expanded obstacle. Any collision between the shrunken and expanded obstacles requires incrementing w_r.

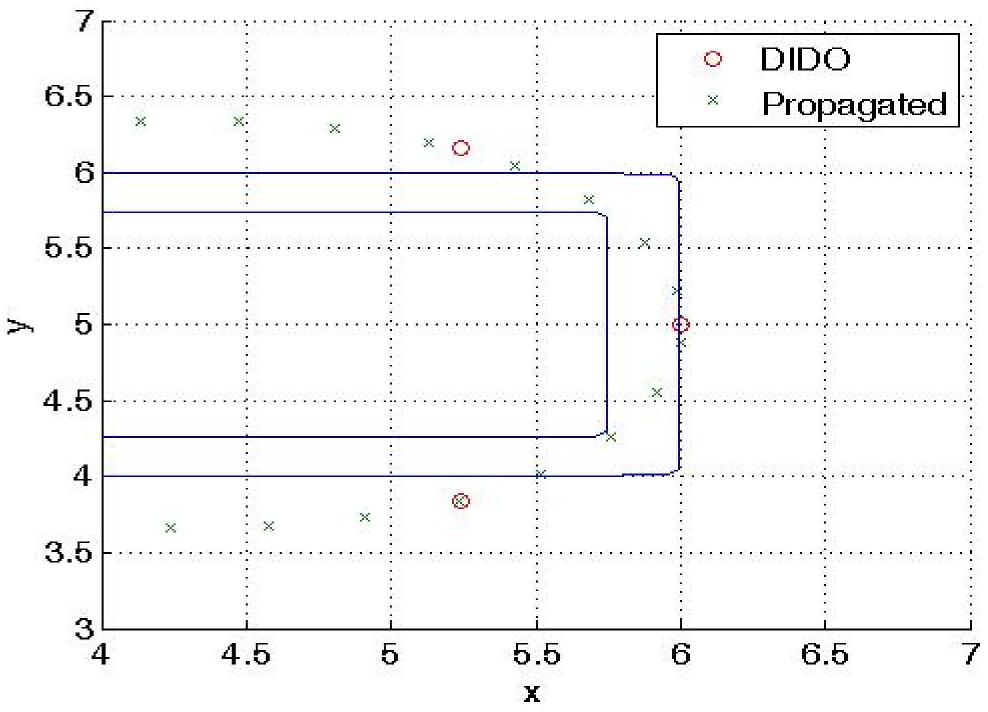


Figure 85. Example of a Grazing Collision.

The idea of incrementing w_r for grazing collisions created three more issues that needed to be addressed. The first of these issues was the fact that r(t) (Equation (50)) becomes lower and lower as *p* in Equation (35) is increased. In other words, the running cost becomes less and less sensitive to w_r as *p* increases. Figure 86 shows a comparison of the running cost vs. distance from an obstacle for varying values of *p*. Simulations have shown that for $p > 4$, the running cost decreases to zero too quickly for w_r to be able to be incremented and have the necessary effect of keeping the vehicle far enough from the obstacle to avoid collision when navigating around the obstacle edges. For this reason, Equation (35) will be modified for use by the running cost of Equation (50) by substituting *q* in the place of *p*. The running cost of all obstacles will use values of $q \leq 4$. This has the effect of accounting for the vehicle's turning dynamics while only modeling its kinematics.

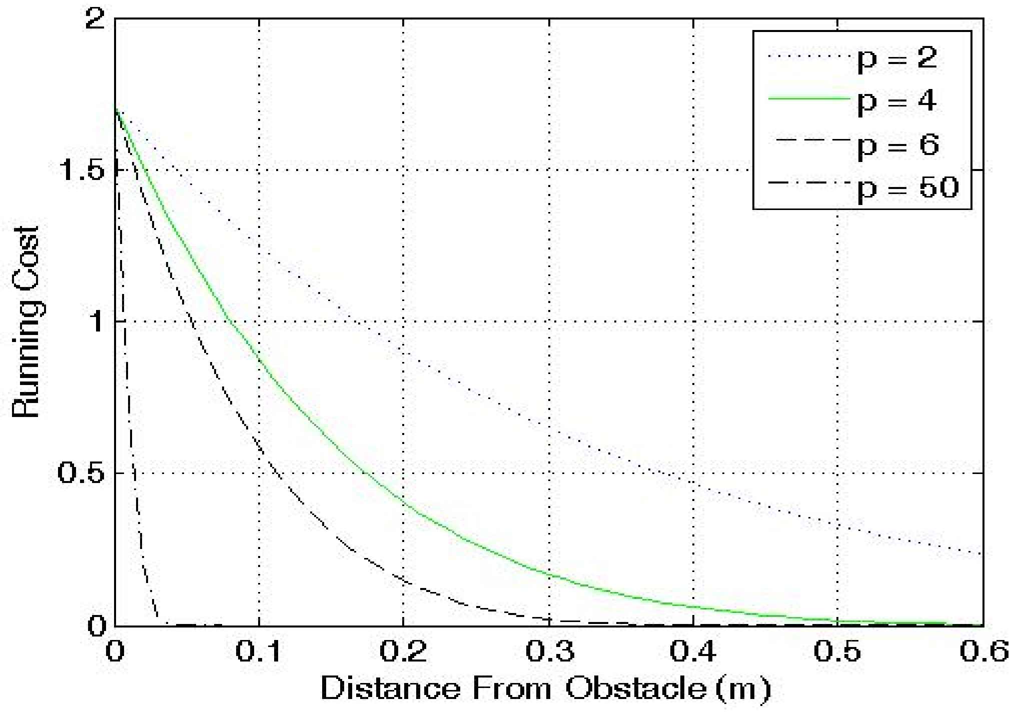


Figure 86. Running Cost vs. Obstacle Distance.

The second issue is how much to increase w_r when a grazing collision is detected and an increment is required. Simulations have shown that w_r need not be as high for lower values of *p* compared to higher values. This was expected, because the higher the value of *p* the sharper the corners on the obstacle; and sharper obstacle corners require more obstacle clearance to successfully navigate. These conclusions show the need to develop an equation that increments w_r as a function of *p*. Furthermore, it was found that the increments of w_r at low values of *p* can be small, but as *p* increases the increments must occur in larger steps to ensure obstacle clearance. There also comes a point when further increases in *p* do not require larger increments in w_r. Equation (55) shows the formula for incrementing w_r. It allows higher increments of w_r as *p* increases, but at a decreasing rate, not to exceed an increment of *0.25*. Figure 87 shows the value of the increments of w_r vs. *p*. Values above *0.25* are not required, because that level is fast enough for any obstacle avoidance problem. The need for an instantaneously high value of w_r (such as in the case of a pop-up obstacle) will be discussed later.

$$\Delta w_r = 0.25\left(1 - e^{-p/8}\right) \tag{55}$$

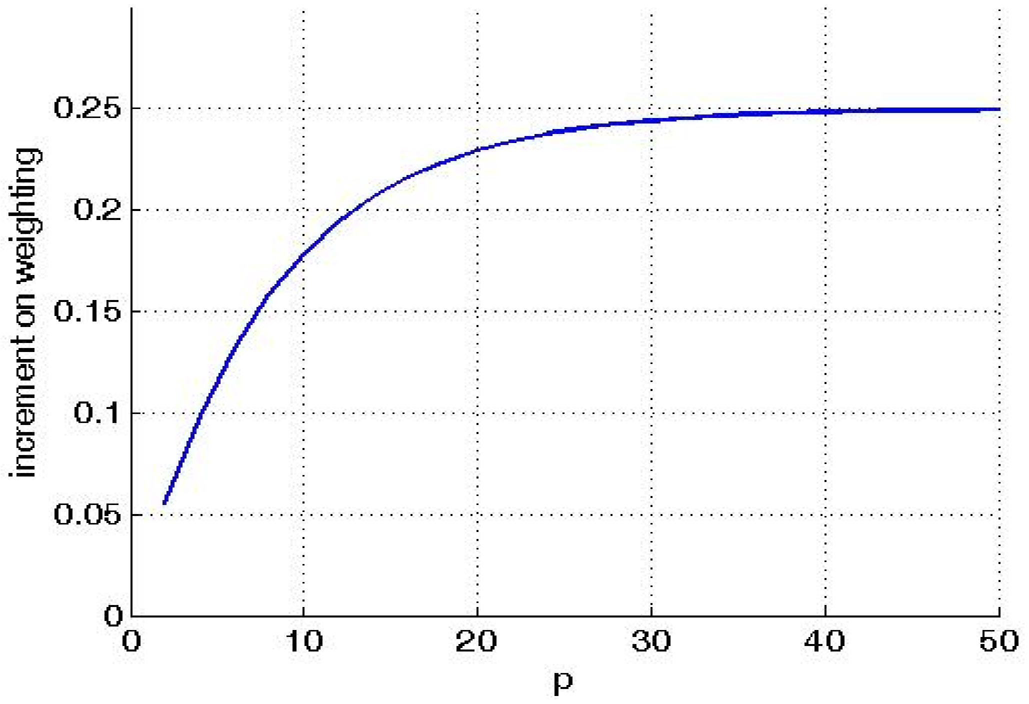


Figure 87. Increment on w_r vs. p.

Figure 88 shows an example (using $p = 10$) of the evolution of the vehicle trajectory as w_r is incremented to ensure clearance of the obstacle. The first window shows the initial trajectory with the initial value of $w_r = 0.15$. The evolution of the trajectory shows that it takes three increments of w_r to ensure obstacle clearance.

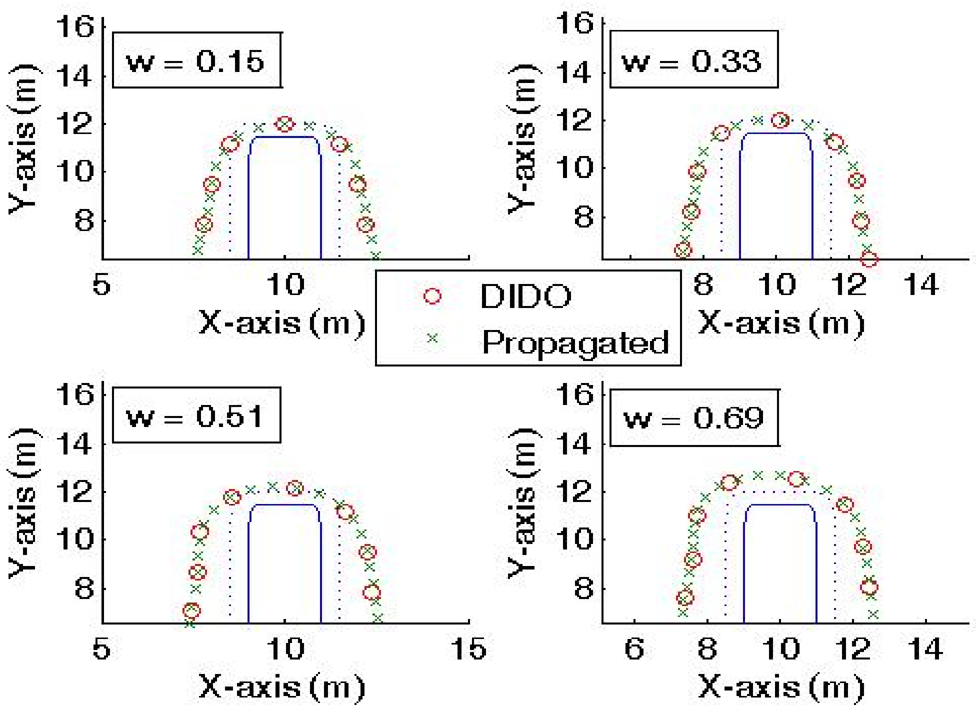


Figure 88. Evolution of Vehicle Trajectory to Clear Obstacle (*p* = *10*).

The scenario of Figure 88 was repeated for various values of *p* to develop Table 4, which allows a comparison of the number of increments and final value of w_r that is reached to successfully navigate the obstacle using Equation (55) to affect the increments. Table 4 validates the use of Equation (55) in that it shows the trajectory to be safe within a reasonable number of increments regardless of the value of *p*. The spread of values of the final w_r shows the algorithm's ability to autonomously arrive at a suitable w_r for obstacle clearance. It was also found that the vehicle could follow the contour of a circular obstacle (*p* = *2*) so closely that sometimes propagation error resulted in a collision of the vehicle with the obstacle within the allowed run time. For this reason, the starting value for w_r when *p* = *2* was adjusted in the algorithm to $w_r = 0.2$.

Table 4. Evolution of w_r for Various Values of *p*.

p	w_r **Increment**	**# of Increments**	w_r **Final Value**
2	0.055	1	0.255
4	0.098	4	0.544
6	0.132	4	0.678
8	0.158	3	0.624
10	0.178	3	0.685
20	0.229	3	0.838
30	0.244	3	0.882
40	0.248	3	0.895
50	0.25	3	0.899

In order to show that the required value of w_r varies from situation to situation, the vehicle was given start and goal positions that required a simpler maneuver around the obstacle of Figure 88 (again using $p = 10$). Figure 89 shows the clear trajectory around the obstacle, requiring only 1 increment for a final $w_r = 0.328$.

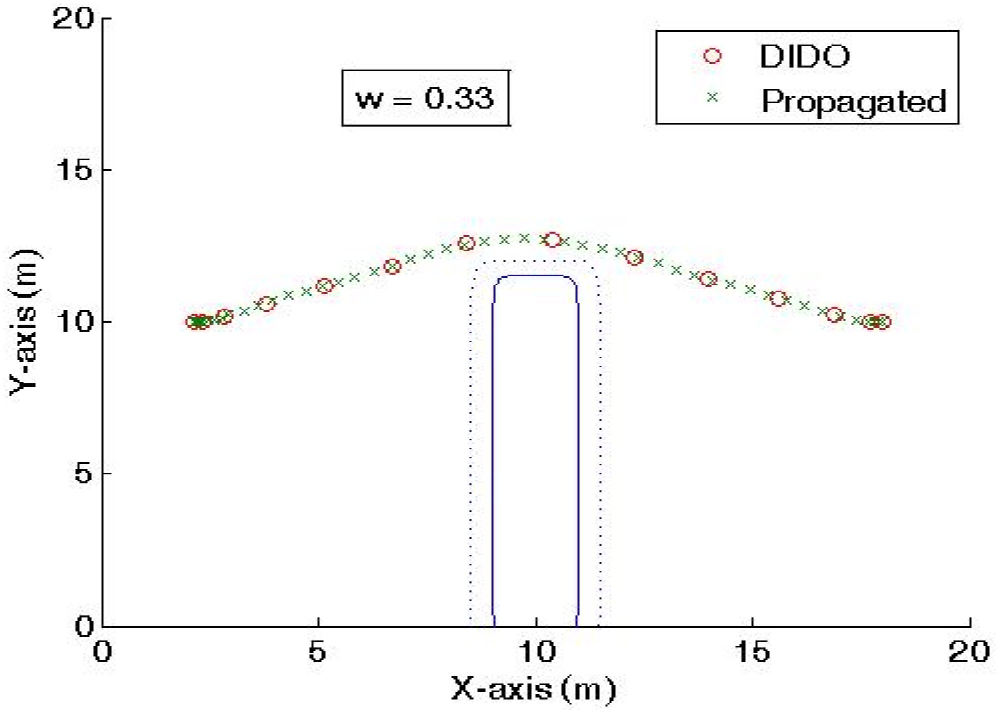


Figure 89. Clear Trajectory for p = 10 and $w_r = 0.33$.

The third and final issue regarding the weighting of the running cost is the fact that there should not be just one w_r that is incremented every time a grazing collision is detected with any obstacle, because it would then be incremented to the worst case obstacle maneuver resulting in a value that is unnecessarily high for avoidance of the other obstacles in the environment. For this reason, we require a separate weighting for each obstacle in the environment. This is a simple modification to the trajectory planning algorithm involving w_r of Equation (51) becoming a vector whose length is equal to the number of obstacles and is pulled into the equation for the running cost. Equation (50) becomes:

$$r(t) = \sum_{i=1}^{n} w_{r_i} \left(e^{e^{-h_i(x(t),y(t))}} - 1 \right) \tag{56}$$

Since w_r is absorbed into Equation (56), the new formula for the overall cost becomes:

$$J[\underline{x}(\cdot),\underline{u}(\cdot),t_f] = t_f + \int_0^{t_f} r(t)dt \qquad (57)$$

Figure 90 shows a safe trajectory for a two-obstacle environment after the running cost weightings were incremented. Each obstacle is labeled with its respective w_{r_i}.

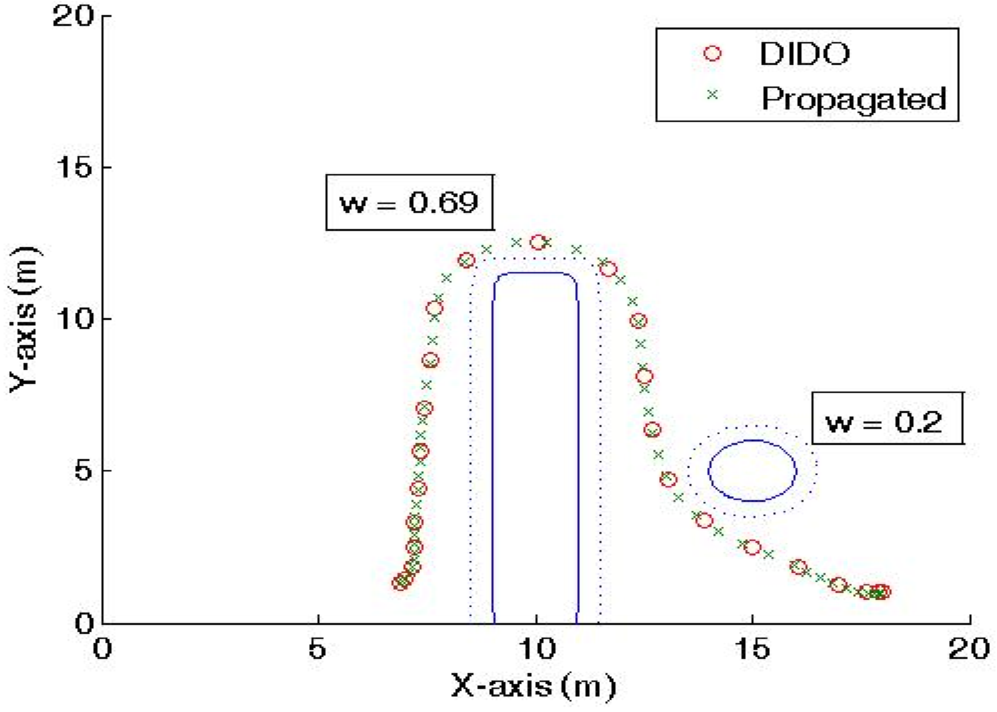


Figure 90. Vehicle Trajectory for a Two-obstacle Environment.

3. Narrow Passages

The concept of navigating narrow passages initially requires the determination of what constitutes a narrow passage. The capability of the trajectory planning algorithm to navigate the vehicle through a narrow passage depends on a number of factors that include: Maximum vehicle speed, minimum vehicle turning radius, run times of the algorithm, weightings of the running cost for the obstacles that make up the narrow passage, shape of the obstacles (value of *p*), nodal spread, and complexity of the trajectory. The maximum vehicle speed and minimum vehicle turning radius are fixed by the design specifications of the vehicle. The algorithm run time varies from run to run (not to exceed the max allowed run time). The running cost weightings are tuned automatically by the algorithm. The obstacle shapes that are input to the problem formulation can be controlled by the operator, but it is desired to define one narrow passage that can be navigated regardless of the obstacle shapes. The complexity of the trajectory is determined by the environment. That leaves only one parameter that can be changed in order to improve the vehicle's ability to navigate through narrow passages: The nodal spread. With that in mind, the trajectory planning algorithm must be modified to first determine if the trajectory traverses a narrow passage, and if it does, it must improve the resolution of the trajectory by using higher node solutions. Using a higher number of nodes allows the vehicle to more precisely navigate through narrow passages, but at the cost of higher run times. Therefore, the use of a higher nodal density must be terminated as soon as the narrow passage condition clears.

Simulations were performed on the two-obstacle environment shown in Figure 91. The outer range of the definition of a narrow passage was determined by starting obstacle 2 at the distance shown and moving it in closer and closer to obstacle 1 until the trajectory solutions were no longer feasible. This was done for varying values of p from $p = 2$ to $p = 50$. When the obstacle clearance was equal to two vehicle widths (200% of vehicle size) or less, solutions were no longer feasible. Therefore, any passage of two vehicle widths or less must be negotiated using higher node solutions. The same experimentation was performed using higher node solutions to determine how narrow a passage could be before solutions became infeasible again. When the clearance between obstacles was 1.3 vehicle widths or less (130% of vehicle size) the higher node solutions started to fail as well. The end result is to define a narrow passage as any obstacle clearance of 130% to 200% of the vehicle size.

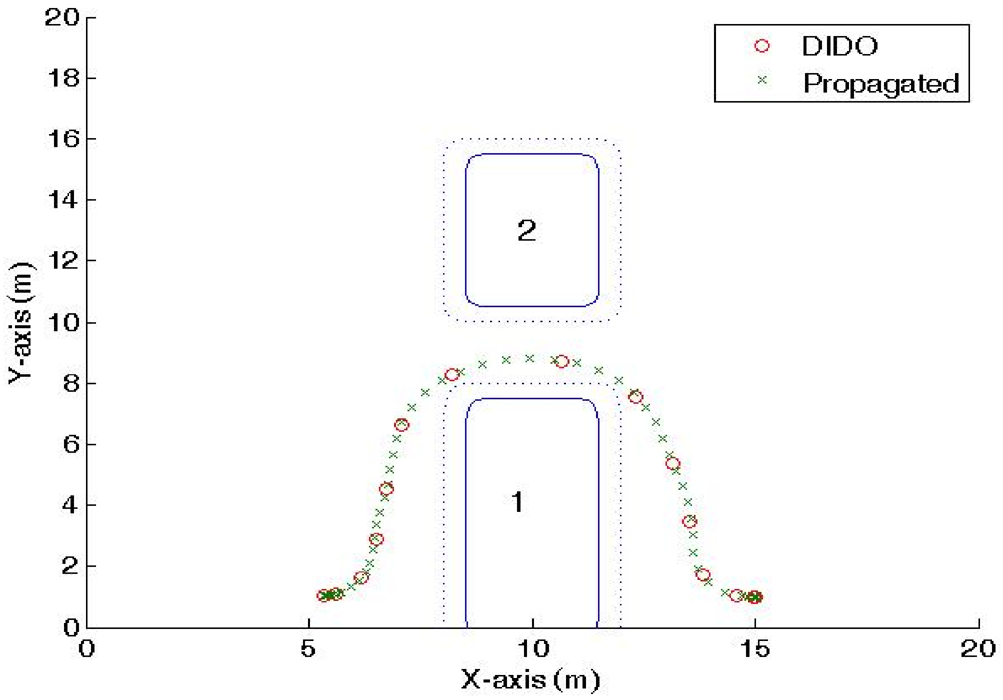


Figure 91. Scenario used for Determining Narrow Passages.

The trajectory planning algorithm was modified to perform a tight passage check immediately after the feasibility and solid collision checks. The tight passage check expands all obstacles by 130% of vehicle size and then searches for simultaneous collisions of the propagated path with more than one obstacle, which would indicate a clearance between obstacles that is too tight for the vehicle to traverse. The result of too tight a passage is the same as for a solid collision or an infeasibility; the vehicle must stop and backtrack to its previous position and formulate a new trajectory. If the vehicle passes the feasibility check, the solid collision check and the tight passage check, it then has a feasible and safe path to the goal. The algorithm then must determine if the calculated nodal spread is adequate, or if it needs to use higher node solutions. It does this by expanding all obstacles 200% and again searches for a simultaneous collision, which would indicate a narrow passage condition exists and higher node solutions are required. Once the narrow passage has been traversed, the algorithm reverts back to lower node solutions. This check for higher node solutions brings another issue to light.

Do we want the algorithm to jump to higher node solutions to negotiate a narrow passage when the passage does not actually occur until later in its trajectory? This would result in higher node solutions and longer run times at an unnecessarily early point in the vehicle's trajectory. What we really desire is to have the algorithm jump to higher nodes only when negotiating a narrow passage is imminent.

4. Vehicle Caution Zone

The concept of the vehicle caution zone was born from the desire not to make changes to parameters (such as raising the number of nodes) too early in the vehicle's trajectory. The vehicle need only concern itself with navigating around obstacles that are imminently in its path. This is not to say that the entire trajectory is not checked on each run of the algorithm. In fact, the trajectory planning algorithm performs the feasibility checks, the solid collision checks and the tight passage checks along the entire trajectory on each run of the algorithm (each time step). On each iteration, once the algorithm determines the vehicle has a feasible and safe path along the entire trajectory, it then performs narrow passage and grazing collision checks on those obstacles that are within its caution zone. The caution zone was chosen to be 20 time steps (using the max allowed run time) into the future along the vehicle's trajectory. Given the max allowed run time of *0.4 sec* and the max vehicle speed of *1 m/s*, the caution zone extends a maximum of *8m* out along the vehicle trajectory from its current position.

The scenario of Figure 92 was developed to show the ability of the trajectory planning algorithm to navigate the vehicle through a narrow passage, and to demonstrate the algorithm's performance using the caution zone as the area of interest for conducting narrow passage and grazing collision checks. Despite the use of as high as 40 node solutions, algorithm run times were always below 50 seconds (sufficient for real-time operation). Figure 93 shows snapshots at various milestones in the vehicle's trajectory. Figure 93a shows the first iteration of the algorithm at the beginning of the problem. The DIDO points show a feasible trajectory, but one can see if the points were connected (continuous vice discrete), collisions would occur with both obstacles 1 and 2. Recall that the algorithm autonomously selects the number of nodes while obeying the relation in Equation (54); at this point in the trajectory it was using 40 node solutions. Figure 93b shows the first snapshot that results in detecting a collision with obstacle 1 within the vehicle's caution zone. The algorithm increments w_{r_1} for obstacle 1. Figure 93c is a snapshot of the next iteration (still 40 nodes), which shows the updated trajectory providing a feasible and safe path around obstacle 1. Note that the path around obstacle 2 is still unchanged, because that portion of the trajectory has not yet entered the vehicle's caution zone. Figure 93d shows the first snapshot that results in detecting a collision with obstacle 2 within the vehicle's caution zone. At this point in the trajectory, the algorithm was using 25 node solutions based on Equation (54). The algorithm increments w_{r_2} for obstacle 2, detects the narrow passage, and commences using higher node solutions. A snapshot of the next iteration is shown in Figure 93e, which indicates a feasible and safe trajectory. The purpose of Figure 93f is to show that once the vehicle clears the narrow passage the algorithm autonomously switches to lower node solutions.

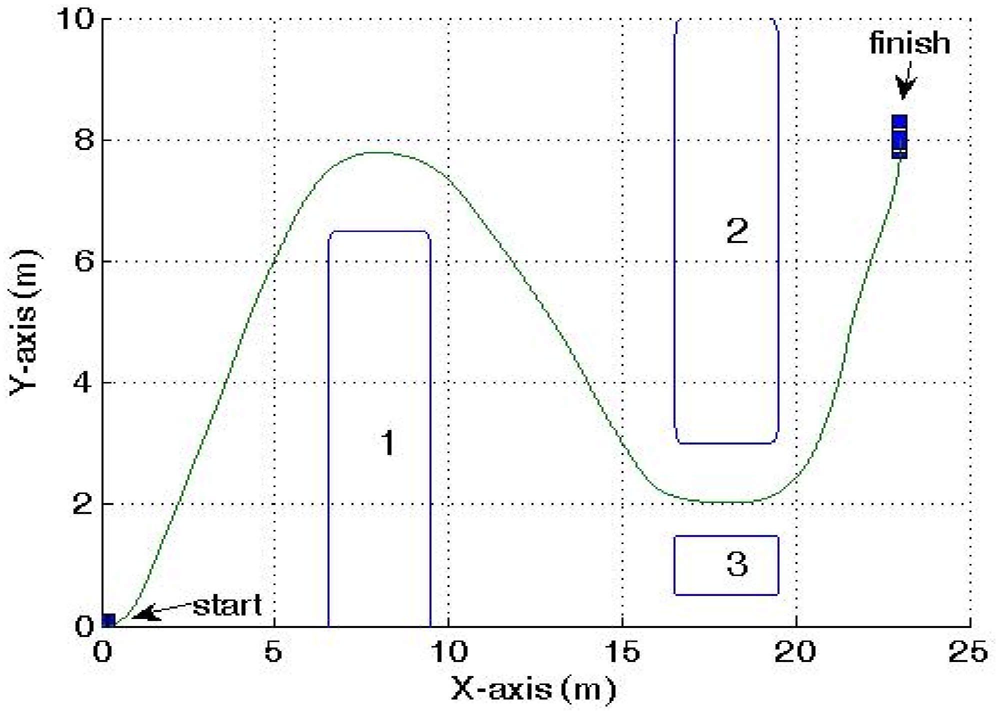


Figure 92. Vehicle Traversing a Narrow Passage.

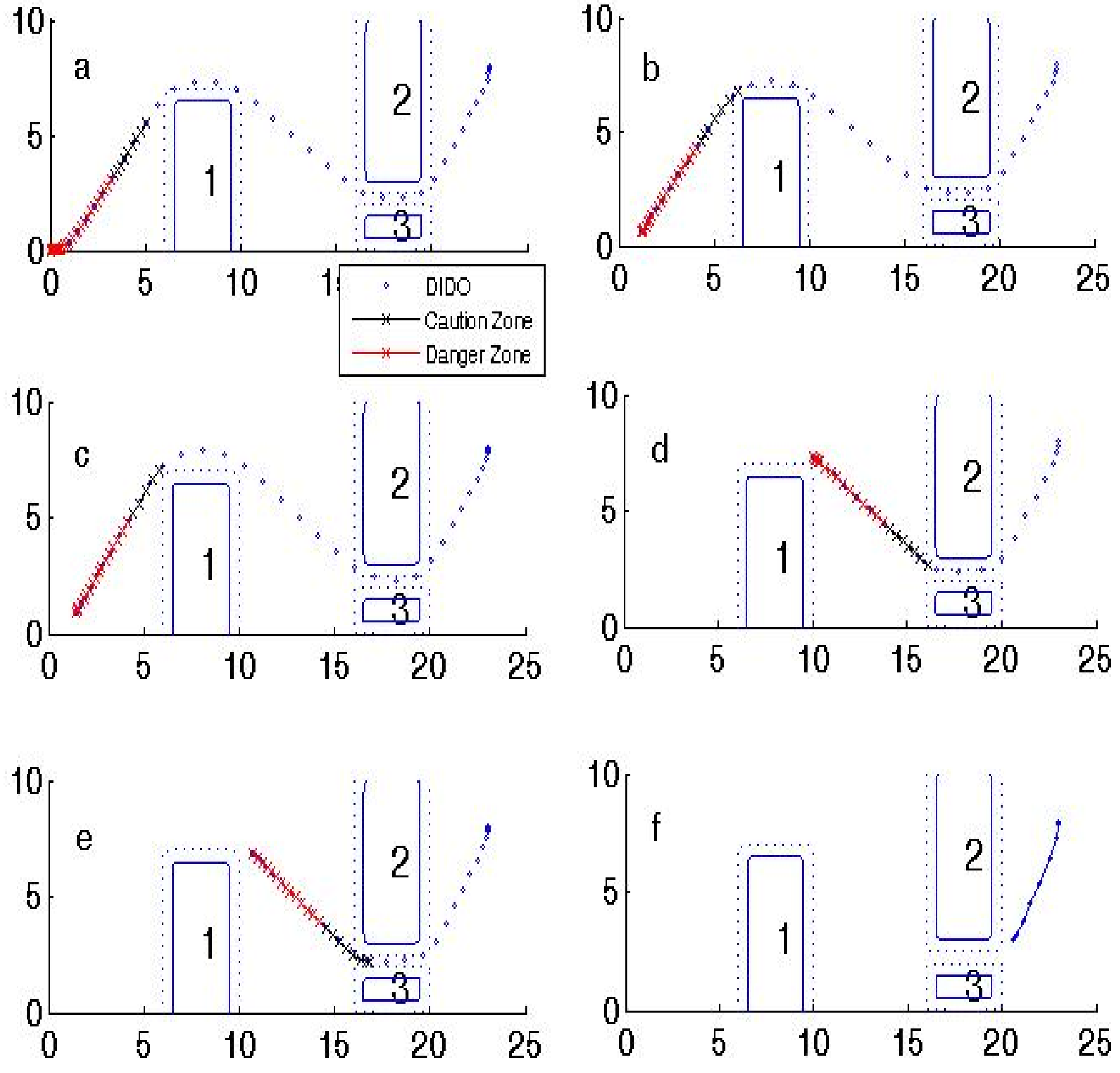


Figure 93. Snapshots of Narrow Passage Problem.

5. Pop-up Obstacles (Vehicle Danger Zone)

What happens if an obstacle pops up very close to the vehicle, precluding the ability to increment the running cost slowly? Table 4 shows it took four time steps to reach the required w_{r_i} in some cases, but if an obstacle pops up too close, the vehicle will need to respond more quickly. The vehicle may require a quick response in other cases as well, such as negotiating a turn that suddenly hits an existing obstacle and requires the vehicle to make a large and quick correction. The danger zone concept was invented for just such an event. If the algorithm detects a collision within the vehicle's danger zone, it raises the weighting on the running cost for that obstacle by $\Delta w_{r_i} = 1$ (more than high enough to clear any obstacle) and switches to high node solutions for better precision. The high value of w_{r_i} and the higher number of nodes will force the vehicle's controls to go to their limits if necessary. If the vehicle still cannot clear the obstacle, an infeasibility or unsafe path will be detected and it will perform actions as required. Once the vehicle clears the obstacle, the algorithm reverts back to lower node solutions. The danger zone is calculated in the same manner as the caution zone and extends out 10 time steps (utilizing the max allowed run time) into the future along the vehicle's trajectory.

The scenario of Figure 94 was developed to show the ability of the trajectory planning algorithm to navigate the vehicle around a pop-up obstacle, and to demonstrate the algorithm's performance using the danger zone concept. The obstacle starts out in position 1 and after 3 seconds it spontaneously repositions to position 2. In position 2, the expanded obstacle is shown in order to display the collision that occurs with the red path, which is the trajectory that the vehicle would have taken if it had not used the danger zone concept. Despite the use of high node solutions once the pop-up obstacle is detected, algorithm run times were always below 50 seconds (sufficient for real-time operation).

Figure 95 shows snapshots at various milestones in the vehicle's trajectory. Figure 95a shows the first iteration of the algorithm at the beginning of the problem. Recall that the algorithm autonomously selects the number of nodes while obeying the relation in Equation (54); at this point in the trajectory it was using 10 node solutions. Figure 95b shows the first snapshot that results in detecting a collision, which happens to occur in the vehicle danger zone. The result is the algorithm performs the increment $\Delta w_{r_i} = 1$ and shifts to high node solutions, as shown by the snapshot of the next iteration (Figure 95c). The purpose of Figure 95d is to show that once the vehicle clears the pop-up obstacle the algorithm autonomously switches back to lower node solutions.

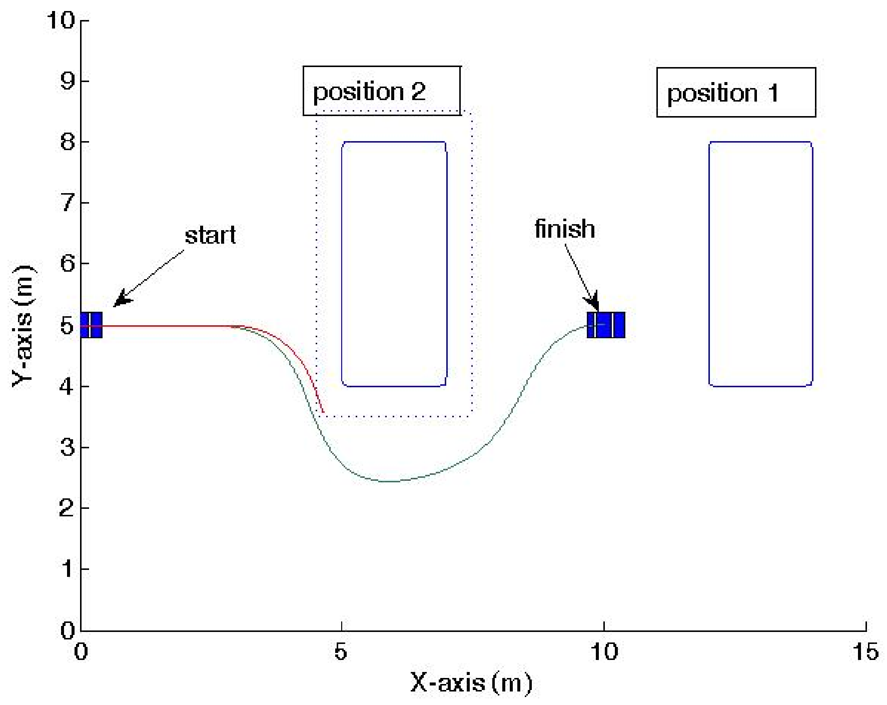


Figure 94. Vehicle Navigating around a Pop-up Obstacle.

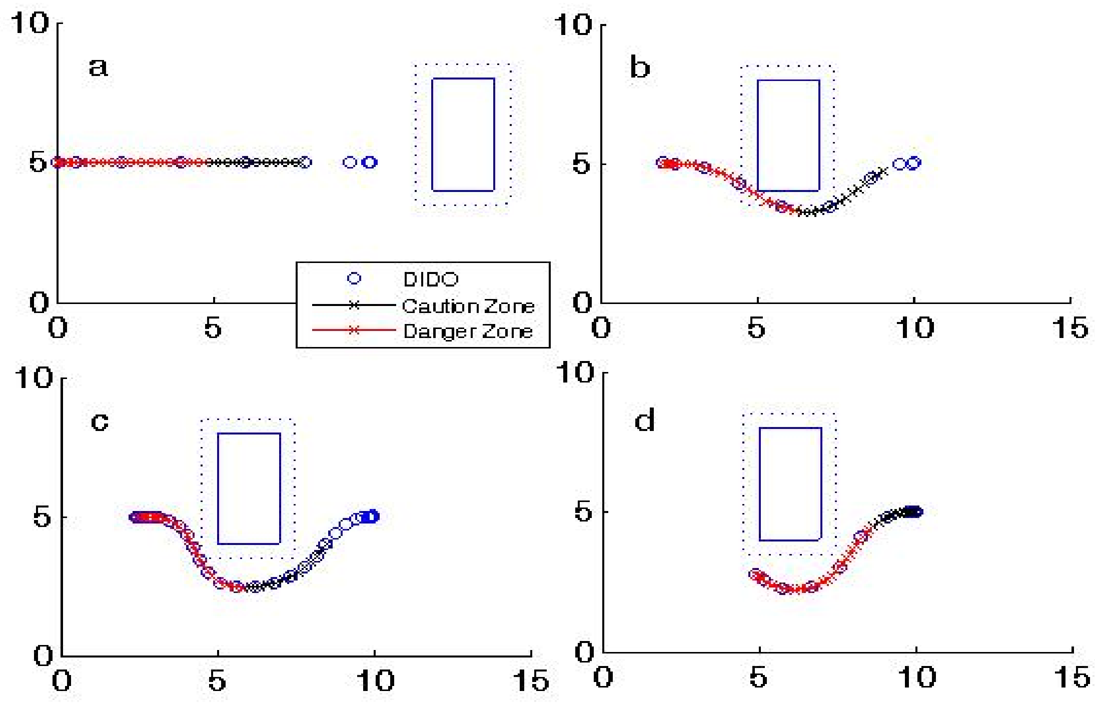


Figure 95. Snapshots of Pop-up Obstacle Scenario.

The scenario of Figure 96 was developed to demonstrate the algorithm's performance using the danger zone concept in a situation not involving a pop-up obstacle. Note the collision with obstacle 2 using the red path, which is the trajectory that the vehicle would have taken if it had not used the danger zone concept. Despite the use of high node solutions once the danger zone actions are initiated, algorithm run times were always below 50 seconds (sufficient for real-time operation).

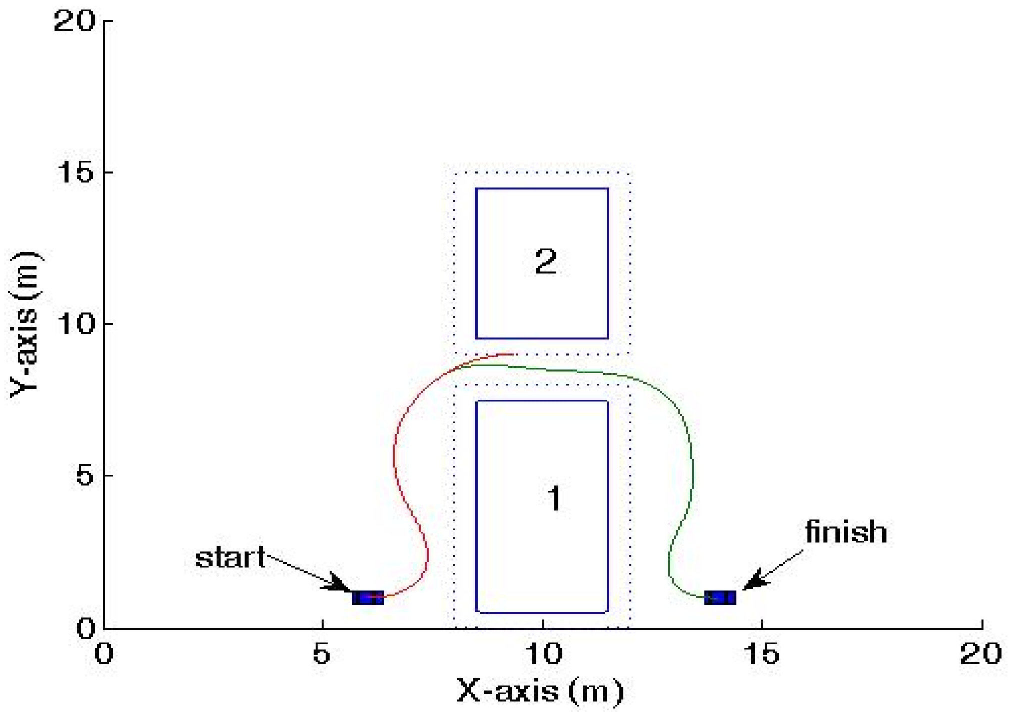


Figure 96. Vehicle Navigation between Obstacles (Not Narrow Passage).

Figure 97 shows snapshots at various milestones in the vehicle's trajectory. Figure 97a shows the first iteration of the algorithm at the beginning of the problem. Recall that the algorithm autonomously selects the number of nodes while obeying the relation in Equation (54); at this point in the trajectory it was using 20 node solutions. Obstacle 1 is detected in the caution zone in this first iteration. Figure 97b shows the first snapshot (now using 15 node solutions) that results in detecting a collision with obstacle 2, which happens to occur in the vehicle danger zone. The result is the algorithm performs the increment $\Delta w_{r_i} = 1$ and shifts to high node solutions, as shown by the snapshot of the next iteration (Figure 97c). The purpose of Figure 97d is to show that once the vehicle clears the danger zone conditions the algorithm autonomously switches back to lower node solutions.

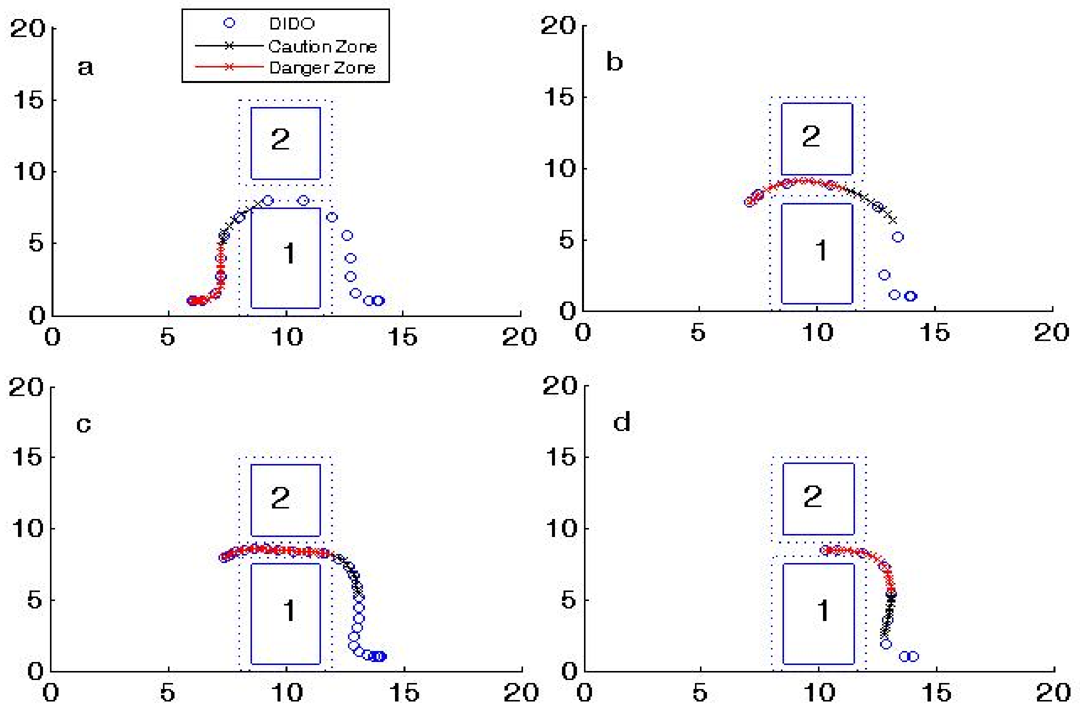


Figure 97. Snapshots of Figure 96 Scenario.

6. The Trajectory Planning Algorithm

A one-line diagram of the trajectory planning algorithm is provided as Figure 98 in order to summarize the flow path. The *Nodal Spread* block determines the required number of nodes based on Equation (54), unless high node solutions are called for based on the narrow passage and grazing collision checks. Once the trajectory passes the feasibility, solid collision and tight passage checks, it can be passed to the vehicle for execution and to the next portion of the algorithm which uses that trajectory as the bias for the next DIDO call. The *Narrow Passage Checks* block determines if high node solutions are needed until the vehicle clears the narrow passage condition. The *Grazing Collision Checks* block makes the determination of which running cost weightings (w_{r_i}), if any, will require tuning prior to the next DIDO call. The tuning increment will be in accordance with Equation (55) unless a first time collision with an obstacle occurs in the vehicle danger zone, resulting in a large increment and high node solutions until the danger zone condition clears.

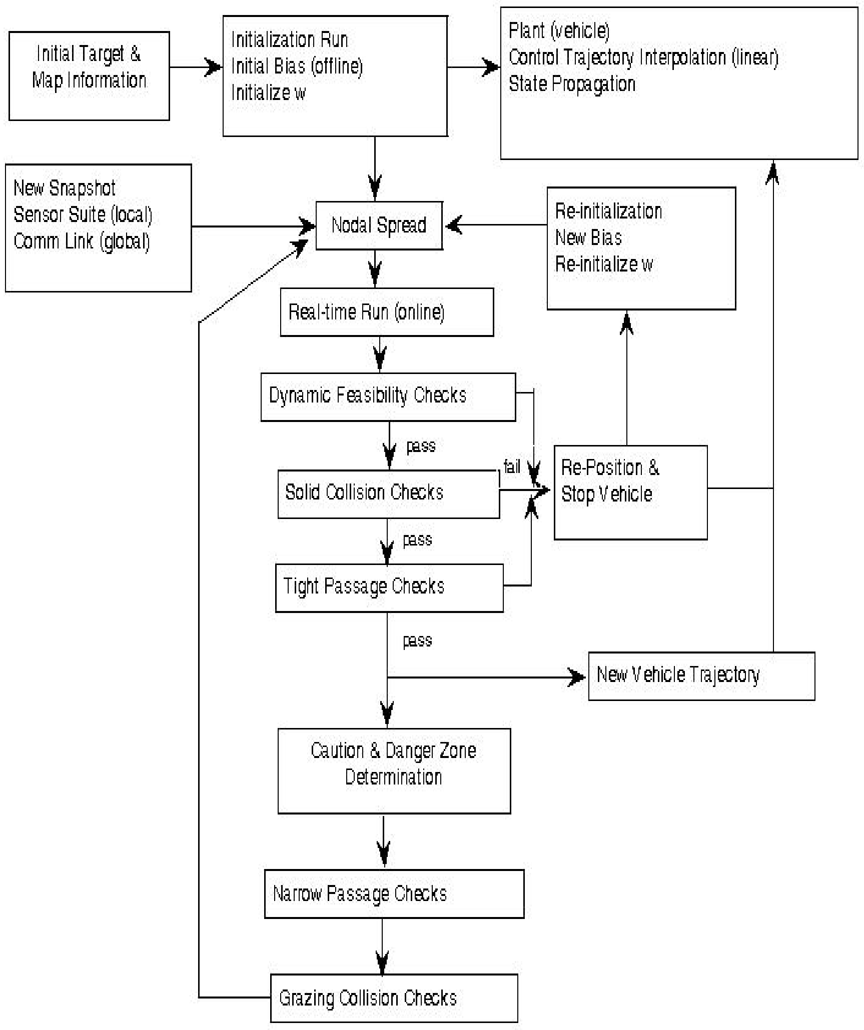


Figure 98. Trajectory Planning Algorithm Flow Path.

V. OBSTACLE AND TARGET MOTION STUDIES

A. BACKGROUND

The purpose of Chapter IV of this work was to develop a trajectory planning algorithm capable of solving any properly formulated problem in any dynamic environment, while meeting the performance criteria set forth in Chapter II. Now that the algorithm has been developed, whether or not it can meet some or all of the basic criteria depends upon the information available and used by the algorithm. The purpose of this chapter is to study obstacle and target motions under varying levels of information. Specifically, we study the algorithm's abilities as an information centric planner to demonstrate the effectiveness of more or less information on the trajectory planning formulation.

B. OBSTACLE MOTION STUDIES

1. The Sliding Door

We start with the familiar sliding door scenario of Figures 99a and 99b, which show the start and end configurations for the obstacle environment respectively. The start and finish positions of the vehicle are as shown. All obstacles are initially motionless. At 3 seconds into the simulation, obstacle 2 moves north at a speed of *0.5 m/s*, resulting in the north passage becoming blocked while opening up the south passage.

The first simulation was completed as before, using snapshots of the environment taken just prior to each iteration of the algorithm, with no prediction of obstacle position. Thus, during the time it takes to generate a new solution, the vehicle maneuvers based on the previous solution with no knowledge of obstacle motion until the next snapshot is taken and the vehicle "senses" the environment has changed. Each run produces a trajectory that solves the static environment problem, even though the environment is dynamic. The resulting trajectory is shown in Figure 100a. The vehicle heads toward the north passage until it is closed off, at which point the vehicle stops, repositions (as required by the algorithm), generates a new bias, and then continues along the new path through the south passage. The trajectory in this case is not perfectly smooth, which can be attributed to the fact that a low node PS-solution results in higher propagation error. Figure 100b shows the smooth trajectory that could be achieved with higher node solutions. Less smooth trajectories are acceptable in trying to keep the number of nodes (and thus the run times) to a minimum.

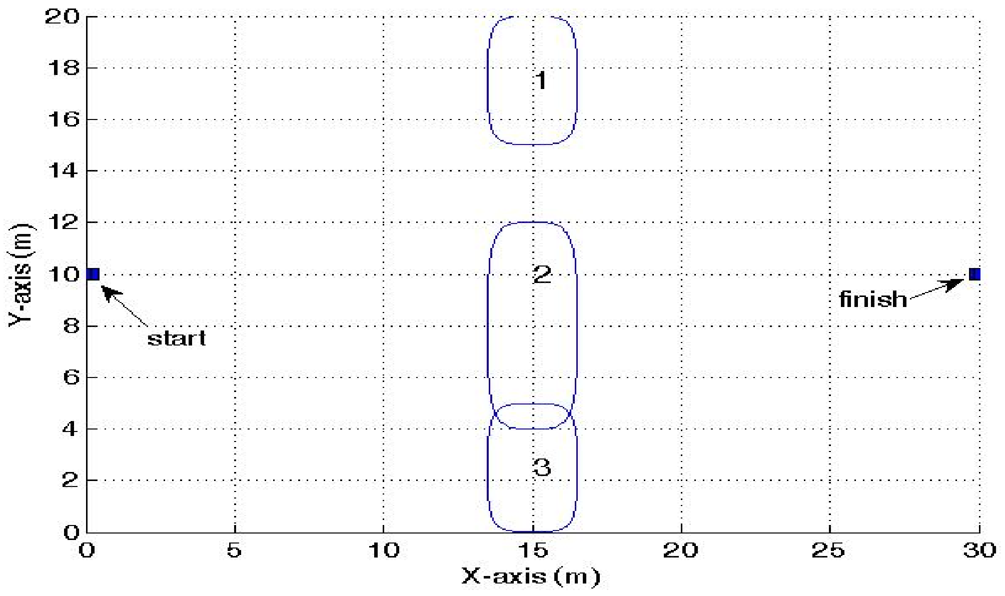


Figure 99a. Start Configuration for Sliding Door Problem.

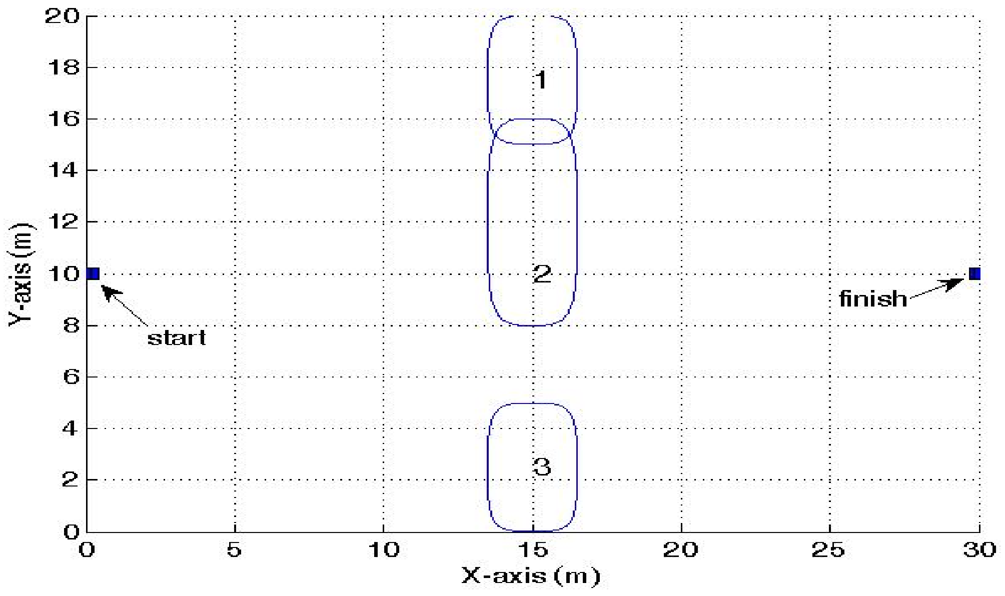


Figure 99b. End Configuration for Sliding Door Problem.

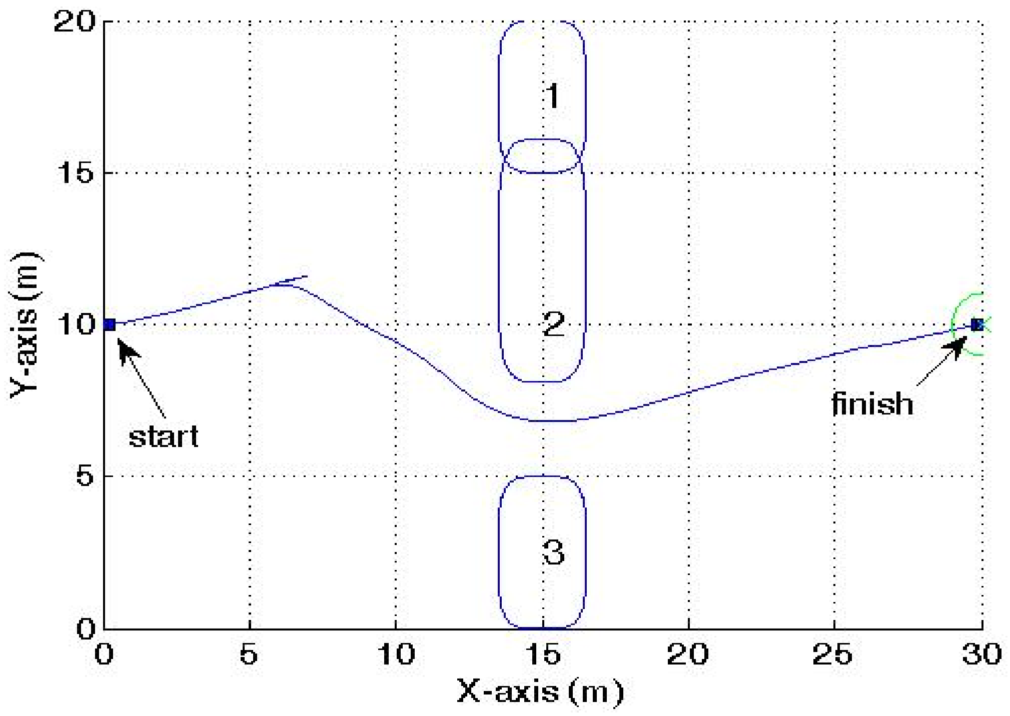


Figure 100a. Trajectory for Sliding Door Problem (Snapshots).

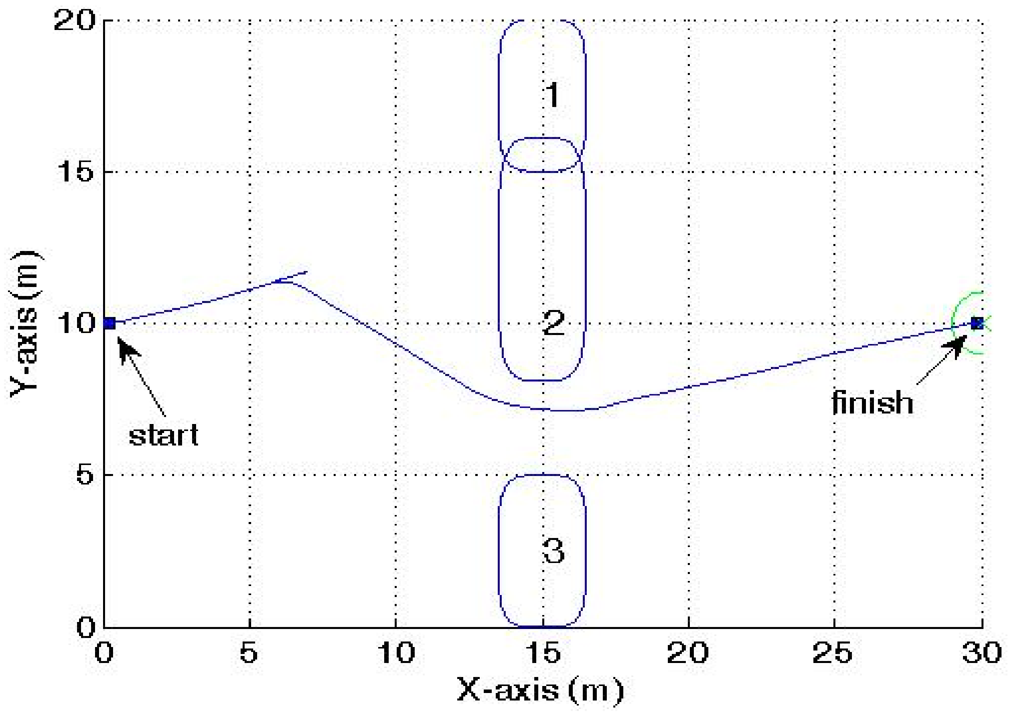


Figure 100b. Smoother Trajectory for Sliding Door Problem (Higher Node Solutions).

The sliding door scenario was repeated with the addition of the obstacle's position prediction using course and speed. This can be achieved simply by comparing successive environment snapshots to estimate a moving obstacle's course and speed, and then using that course and speed to predict the obstacle's future positions (for all time) as part of the trajectory planning problem. Figure 100c shows the resulting trajectory. The vehicle initially heads toward the northern passage. Once obstacle 2 begins to move north, the algorithm determines the obstacle's future position using its course and speed, resulting in the vehicle autonomously changing its course to steer clear of the obstacle. Obstacle 2 then stops moving, resulting in the vehicle making a new course correction further to the south, in order to steer clear. This course correction is necessary because while the obstacle was moving, the prediction of its position was based on the assumption that it would continue on its current course and speed (to infinity). The trajectory planner generates each new trajectory based on the current information on obstacle positions, courses and speeds. It does not attempt to predict course or speed changes.

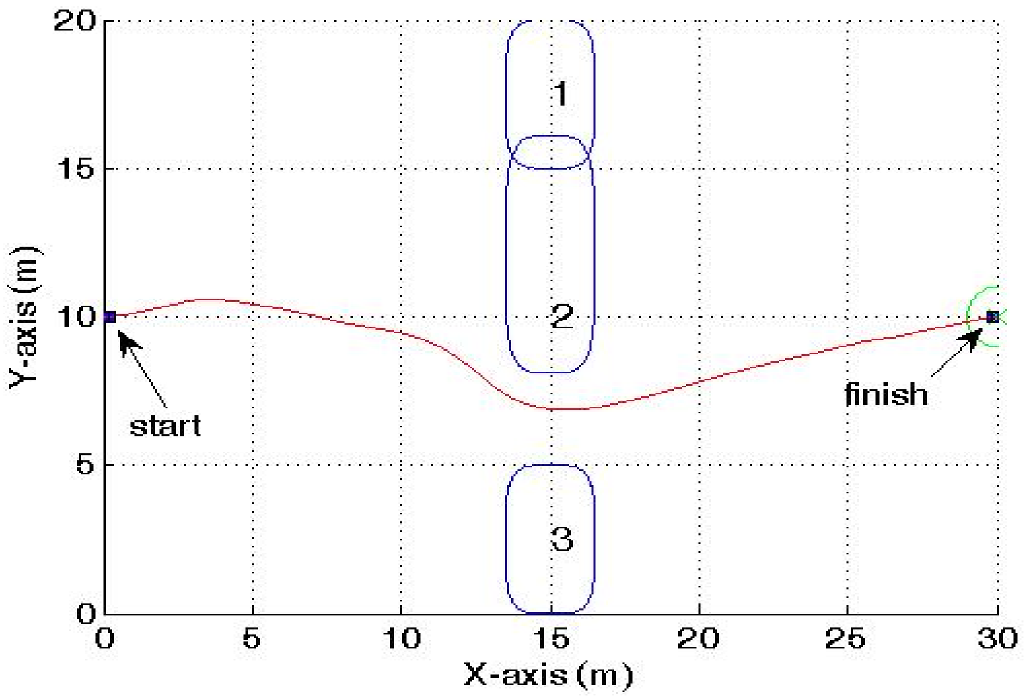


Figure 100c. Sliding Door Problem (Prediction using Course and Speed).

Finally, the sliding door scenario was repeated with complete *a priori* knowledge of obstacle 2 motion; i.e., all future course and speed changes are known in advance. Figure 100d shows the resulting trajectory. The vehicle, having complete knowledge of the future movement of obstacle 2, simply heads immediately down the optimal path through the south passage.

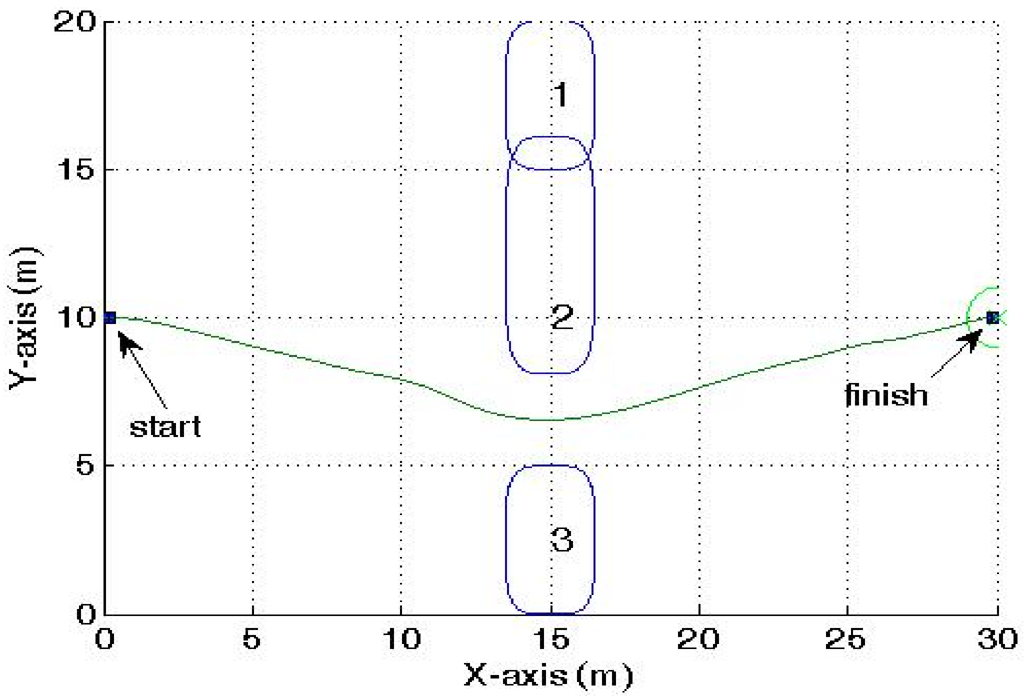


Figure 100d. Sliding Door Problem (Complete *a priori* Knowledge).

The three sliding door scenarios are plotted together on Figure 100e. The scenarios using no prediction and using course and speed for prediction start on the same trajectory, because obstacle 2 does not start moving until the three-second point in the simulations; and without motion, the two scenarios are identical. Once motion begins, using prediction results in a trajectory that is closer to the time-optimal solution based on complete information. When the obstacle position was not predicted (snapshots), the maneuver time was 41.3 seconds, which did not include the extra time it took to calculate a new southerly bias when the north passage became blocked. When prediction was used, the maneuver time was 33.5 seconds. With complete *a priori* knowledge of the environment, maneuver time was slightly shorter at 33 seconds. Thus, the algorithm performs better as more information is provided to it. Note also that the trajectories converge to the same solution once the vehicle is past the obstacles. This is because no new information is available to distinguish one trajectory from the other.

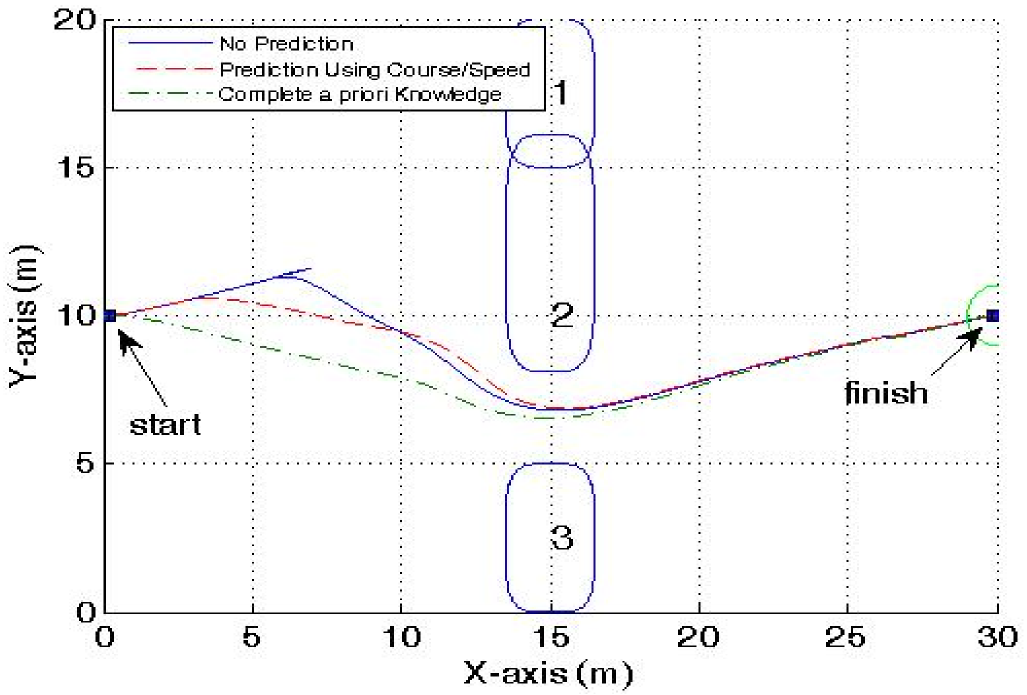


Figure 100e. Summary of Sliding Door Scenarios.

2. The Cyclic Sliding Door

The sliding door scenario is modified so that the door continuously slides back and forth, thus alternately blocking the north and south passages. This type of problem provides the opportunity to examine the trajectory planning algorithm's performance at all three information levels (i.e., snapshots (no prediction), prediction, and *a priori* knowledge). It allows us to create success or failure in algorithm performance simply by adjusting the cycling speed of obstacle 2 (i.e., raising and lowering its frequency). It should be noted at the outset that given the maximum vehicle speed of *1 m/s* and the obstacle widths of *3m* (not including their expansion to account for vehicle size), any obstacle 2 speed greater than *0.6 m/s* will result in failure, because the vehicle cannot traverse through the opening quickly enough. In other words, above a cycling speed of *0.6 m/s* there can be no solution to the problem due to the physical limit on the vehicle speed; i.e., no amount of autonomy can overcome physics.

We start at the lowest information level (i.e., snapshots) and the slowest cycling speed (obstacle 2 moves at *0.1 m/s*). In this scenario obstacle 2 moves so slowly that it never has a chance to reverse course and head south again. In other words, it does not even complete a half cycle. The resulting trajectory is shown in Figure 101a. The position of obstacle 2 in Figure 101a corresponds to its final position at the end of the scenario, and does not indicate its position at the time the vehicle passed through the north passage. Figure 101b shows the evolution of the trajectory at 4 instances in time. Obstacle 2's movement is slow enough that the vehicle need not change course to traverse the passage. The maneuver time for this scenario was 33.5 sec. Future figures showing the full vehicle trajectory from start to finish will indicate obstacle positions at the final time only (as in Figure 101a). Trajectory evolutions in time (as in Figure 101b) will be given where appropriate.

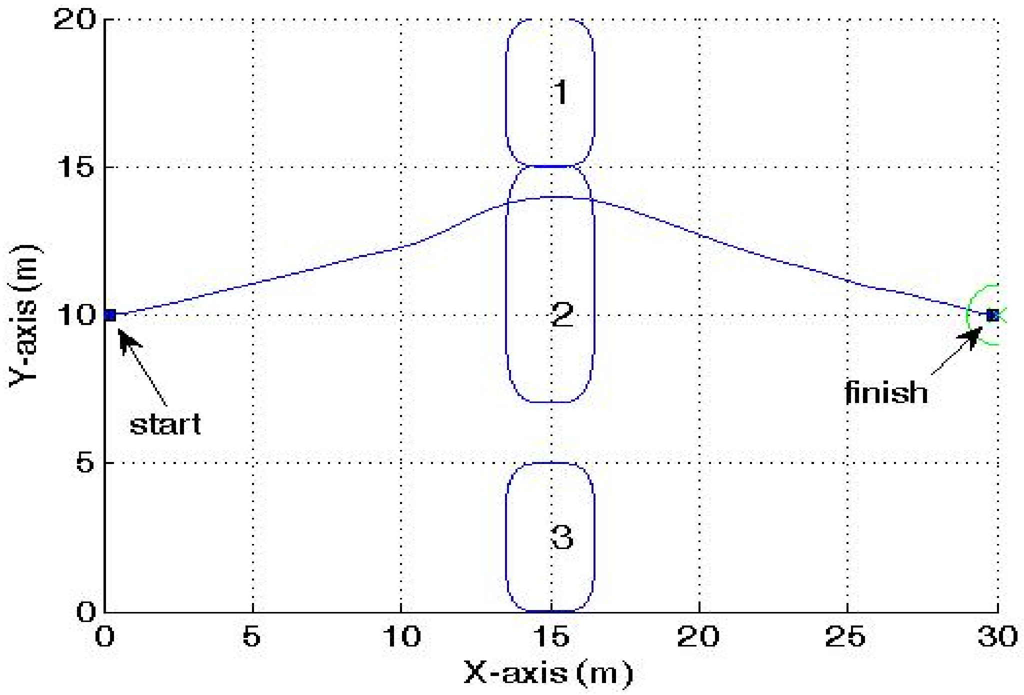


Figure 101a. Cyclic Sliding Door (Snapshots // *0.1 m/s* cycle speed).

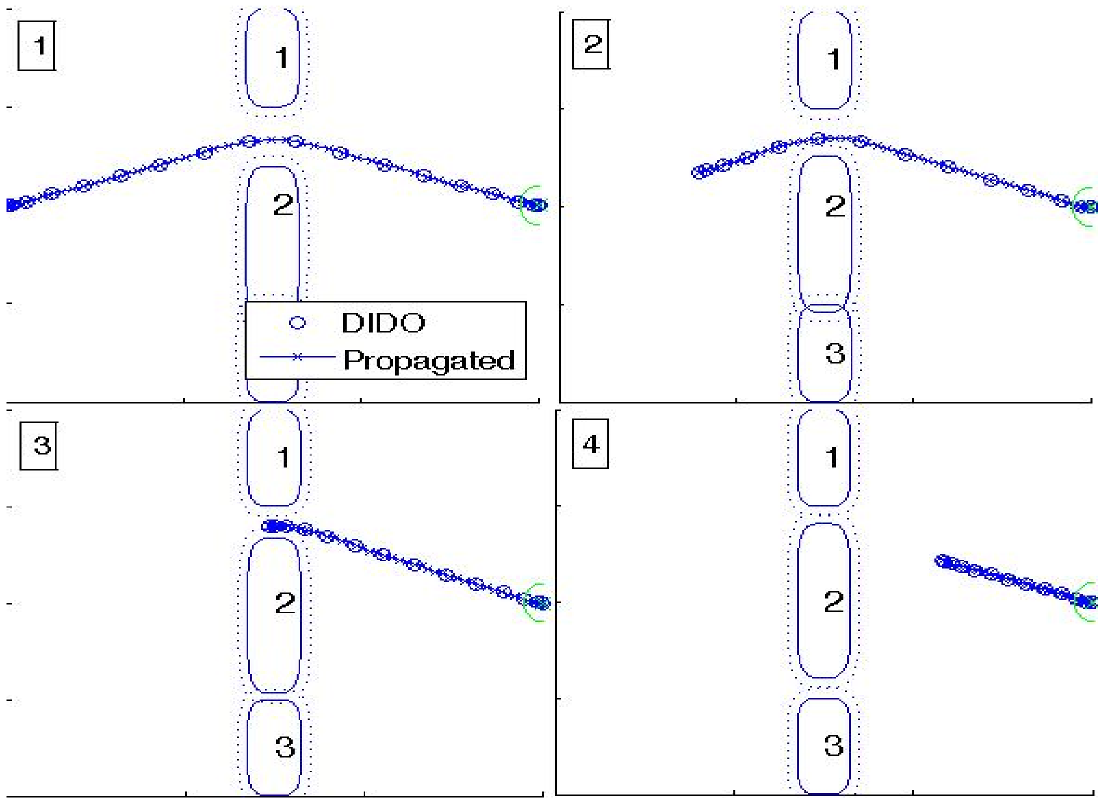


Figure 101b. Evolution of Cyclic Sliding Door (Snapshots // *0.1 m/s* cycle speed).

The cycle speed was then raised to *0.2 m/s*, which prevented the vehicle from reaching the north passage in time to pass through. Since prediction is not being used, the vehicle does not know the north passage will be closed off until it actually happens, requiring the vehicle to stop, reposition, and reformulate a new bias. Figure 102 shows the resulting collision. The obstacle is moving slowly enough to infer a solution exists, but without prediction the vehicle cannot pass. The algorithm also failed to safely guide the vehicle through the passage (north or south) for all other speeds up to the *0.6 m/s* limit. We can suffice it to say that without more information the algorithm cannot safely guide the vehicle through the cyclic sliding door example. This scenario shows a major weakness when conducting trajectory planning using snapshots in a dynamic environment; specifically, when the vehicle is traveling in close proximity to a moving obstacle, the movement of the obstacle between snapshots can cause a collision condition.

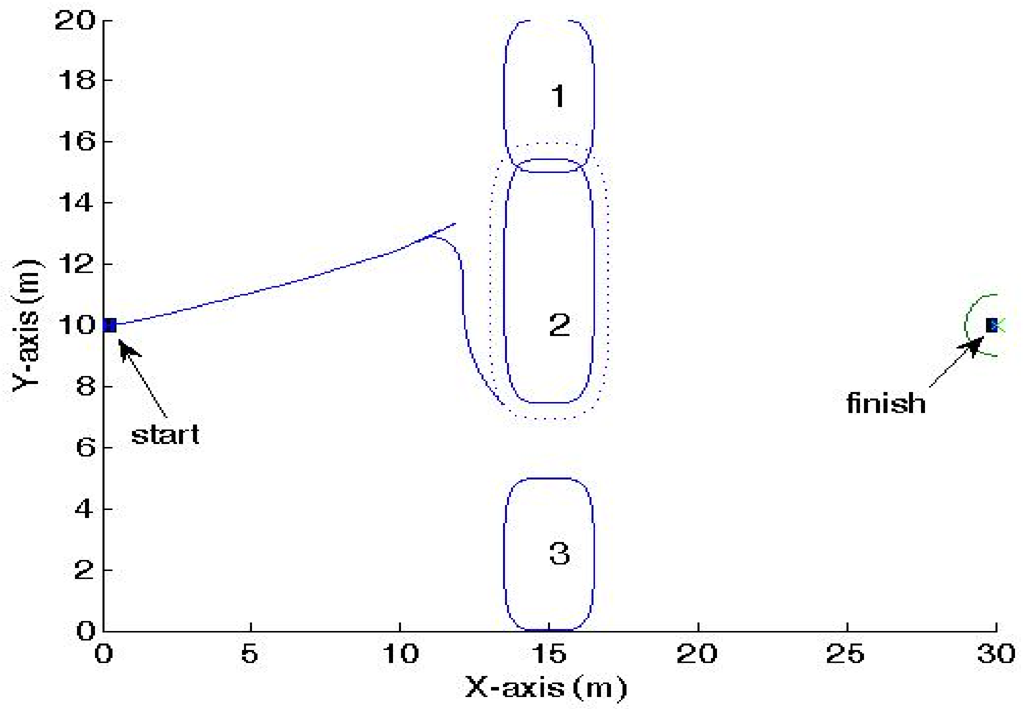


Figure 102. Cyclic Sliding Door (Snapshots // *0.2 m/s* cycle speed).

The same scenarios presented above in Figures 101 and 102 were simulated again using obstacle position prediction from course and speed data. Figure 103a shows the trajectory when cycle speed is at *0.1 m/s*. The maneuver time was *33.4 sec*. The obstacle speed is too slow to show any significant improvement of the trajectory over the case of not utilizing course and speed data (snapshots). The significance of again showing the evolution of the trajectory (Figure 103b) is in the change of the trajectory in the second frame of the figure. We see the trajectory curve further away from the obstacle in frame 2, because the algorithm is predicting the position of obstacle 2 as being further north at the time when the vehicle traverses the north passage.

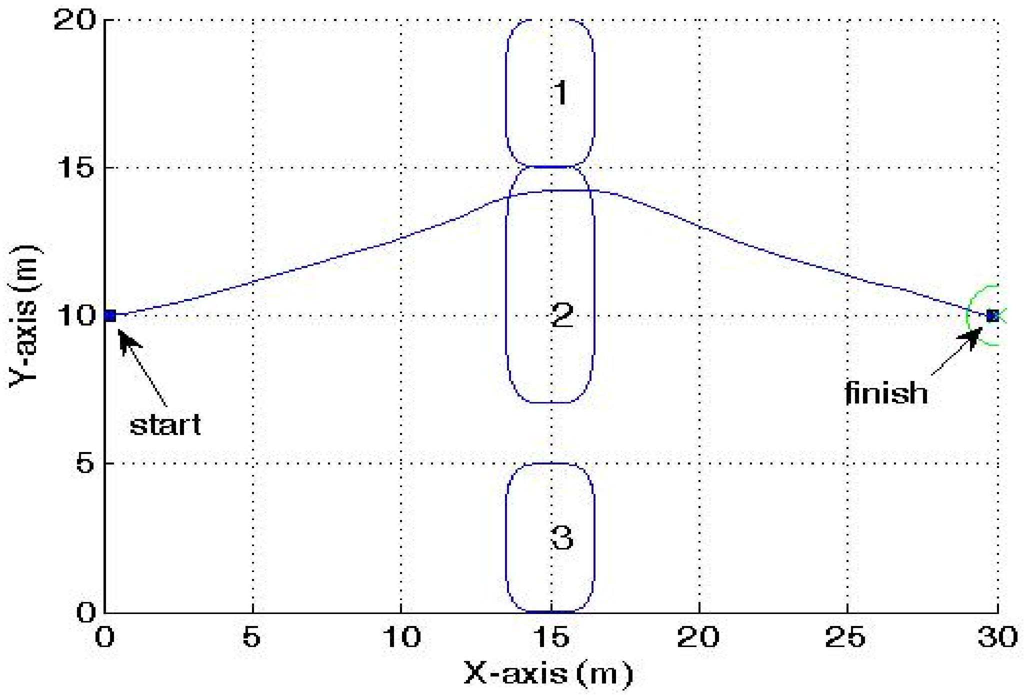


Figure 103a. Cyclic Sliding Door (Prediction // *0.1 m/s* cycle speed).

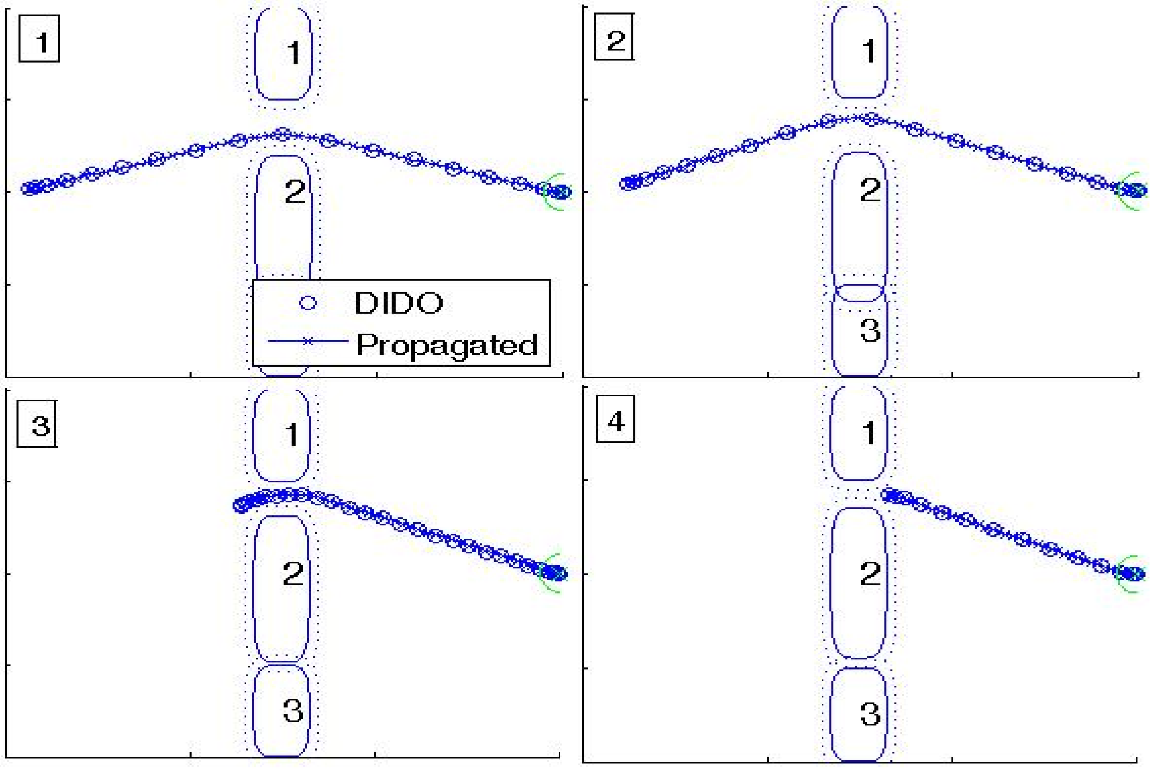


Figure 103b. Evolution of Cyclic Sliding Door (Prediction // *0.1 m/s* cycle speed).

The cycle speed was again raised to *0.2 m/s* to show that with the addition of prediction using course and speed, what was previously a trajectory planning failure became a success, as shown in Figure 104a. The vehicle initially heads to the north passage. Once obstacle 2 begins moving, the algorithm predicts an infeasibility, because the intercept time of the vehicle with either passage results in complete blockage of both passages (obstacle 2 (when expanded to account for vehicle size) completely blocks the path). The result is the vehicle stops, repositions, and reformulates a new bias. In the time it takes the vehicle to stop and reposition, the intercept time changes, and a solution then exists to allow the new bias to be generated. The total maneuver time in this case was *40 sec*. The reason for the large increase in maneuver time is the fact that the vehicle stops and repositions. It is obvious that rather than stop and reposition, the vehicle could simply change its speed slightly to affect a different intercept time that would result in an open south passage; however, it is counter to the minimum time formulation of the optimal control problem to slow down (use less control effort) in order to achieve a feasible path; therefore, its output indicates an infeasible trajectory using the maximum control effort. Figure 104b shows the evolution of the vehicle trajectory; notice the infeasibility in frame 2 of said figure.

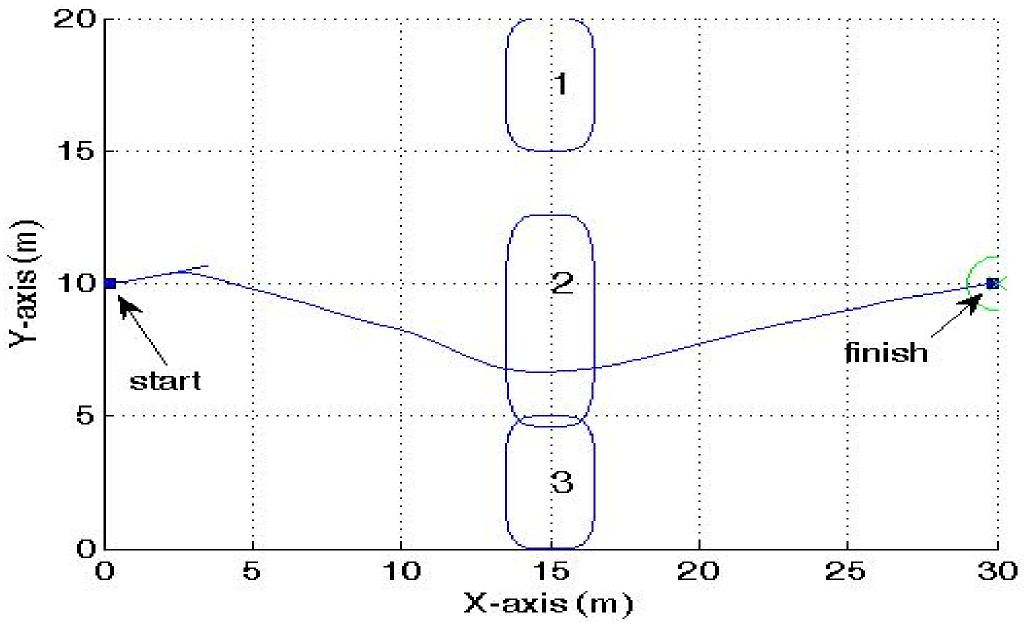


Figure 104a. Cyclic Sliding Door (Prediction // *0.2 m/s* cycle speed).

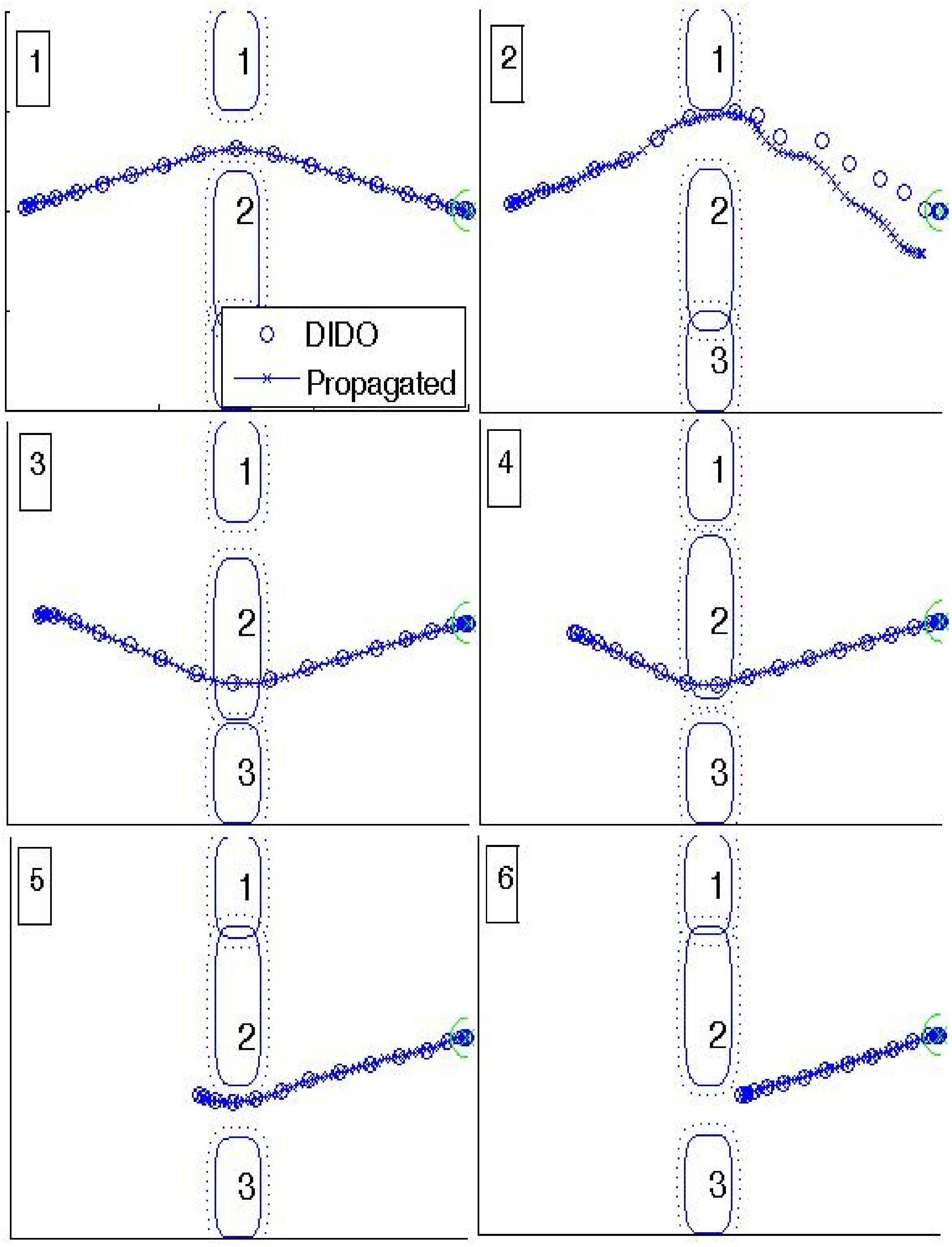


Figure 104b. Evolution of Cyclic Sliding Door (Prediction // *0.2 m/s* cycle speed).

The speed of obstacle 2 was further raised in *0.1 m/s* increments to determine where the algorithm would fail using course and speed to predict obstacle position. Figure 105a represents the trajectory corresponding to the fastest cycling speed (*0.5 m/s*) that the algorithm could handle. The total maneuver time was 36.3 sec, which was faster than in Figure 104, because the vehicle did not have to stop and reposition. Figure 105b represents the evolution of the vehicle trajectory. Frame 1 shows the trajectory prior to the commencement of obstacle 2 movement. In frames 2 and 3, obstacle 2 is moving north, resulting in the prediction that the trajectory could wrap around the south side of the obstacle as it travels north. In frames 4 and 5, obstacle 2 has changed directions to head south, which results in a spontaneous replanning of the vehicle trajectory to wrap around the north side of the obstacle. In frame 6, obstacle 2 has changed directions again and is headed north, but by that point the vehicle has already safely navigated the north passage. Clearly, if the algorithm can predict obstacle position using course and speed, it is more effective than simply using still snapshots of the environment.

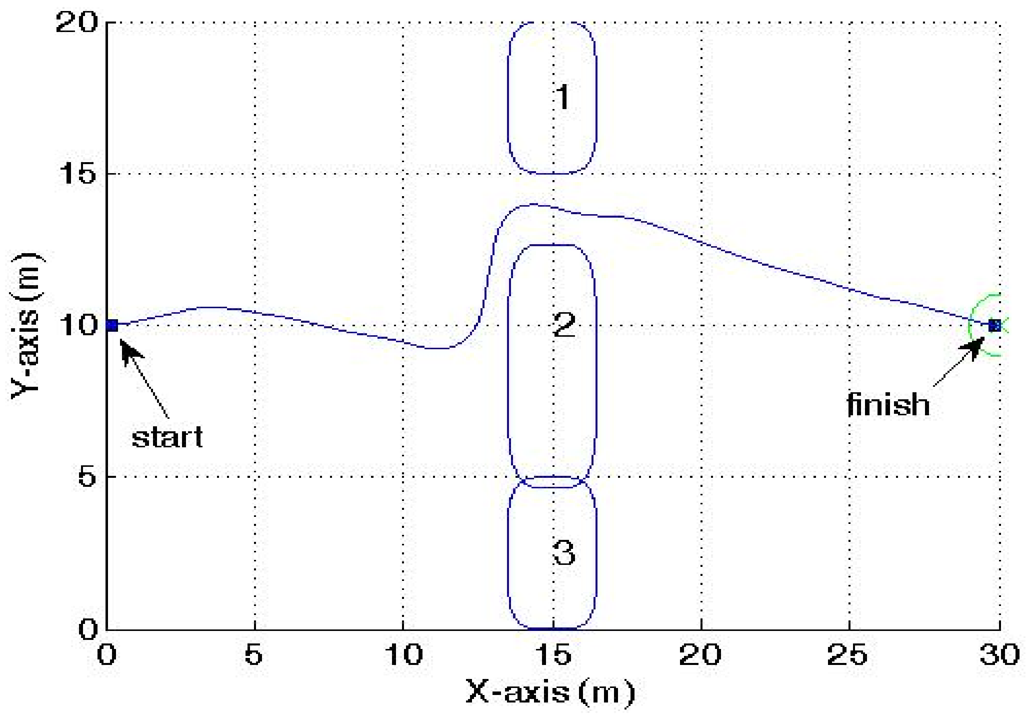


Figure 105a. Cyclic Sliding Door (Prediction // *0.5 m/s* cycle speed).

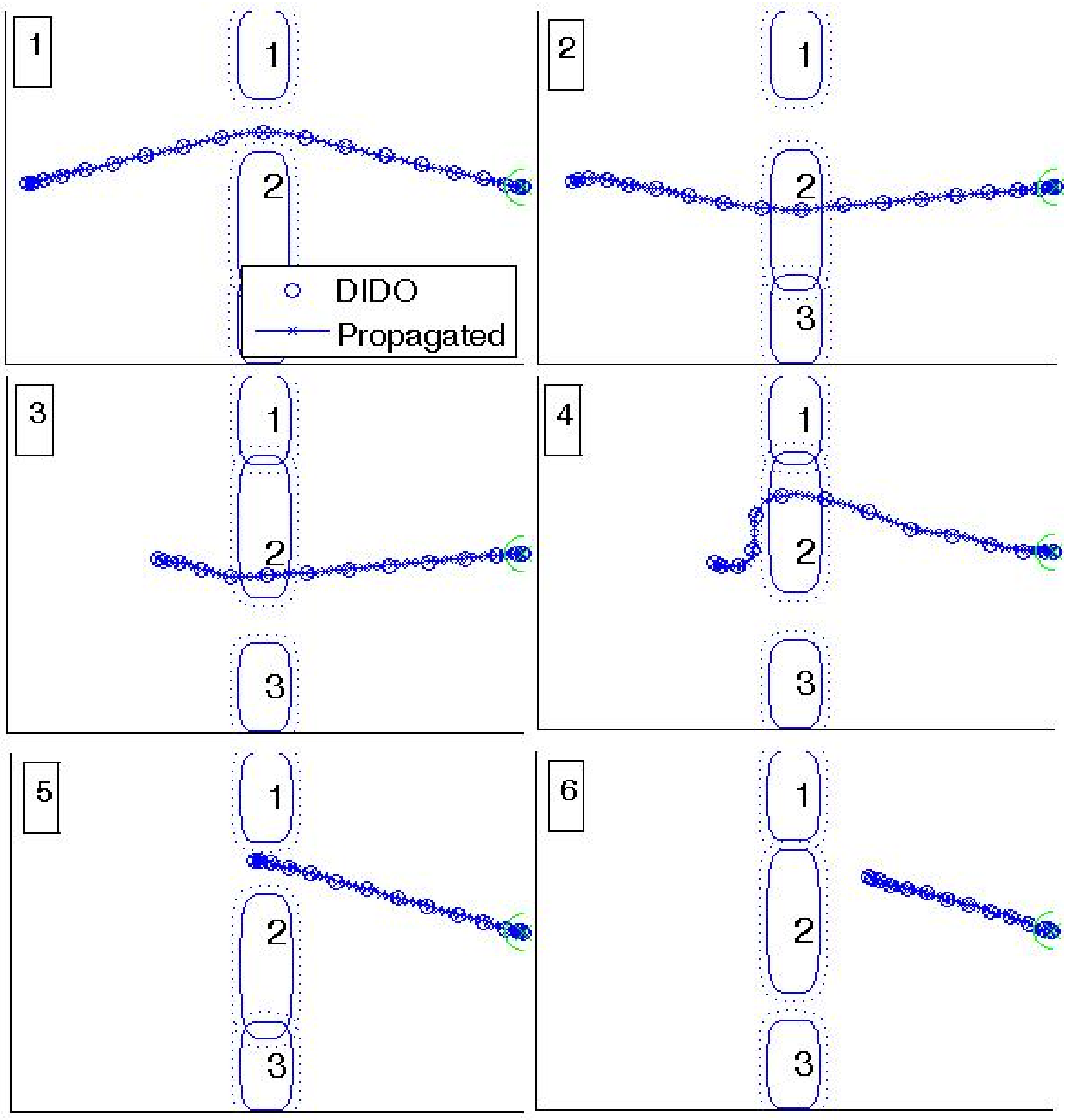


Figure 105b. Evolution of Cyclic Sliding Door (Prediction // *0.5 m/s* cycle speed).

The next logical step here is to show the failure of the algorithm when the speed of obstacle 2 is raised to *0.6 m/s*. Figure 106 represents that trajectory, which changes direction (by prediction using course and speed) as the direction of obstacle motion changes, but the obstacle is too fast for this method to be successful, resulting in collision.

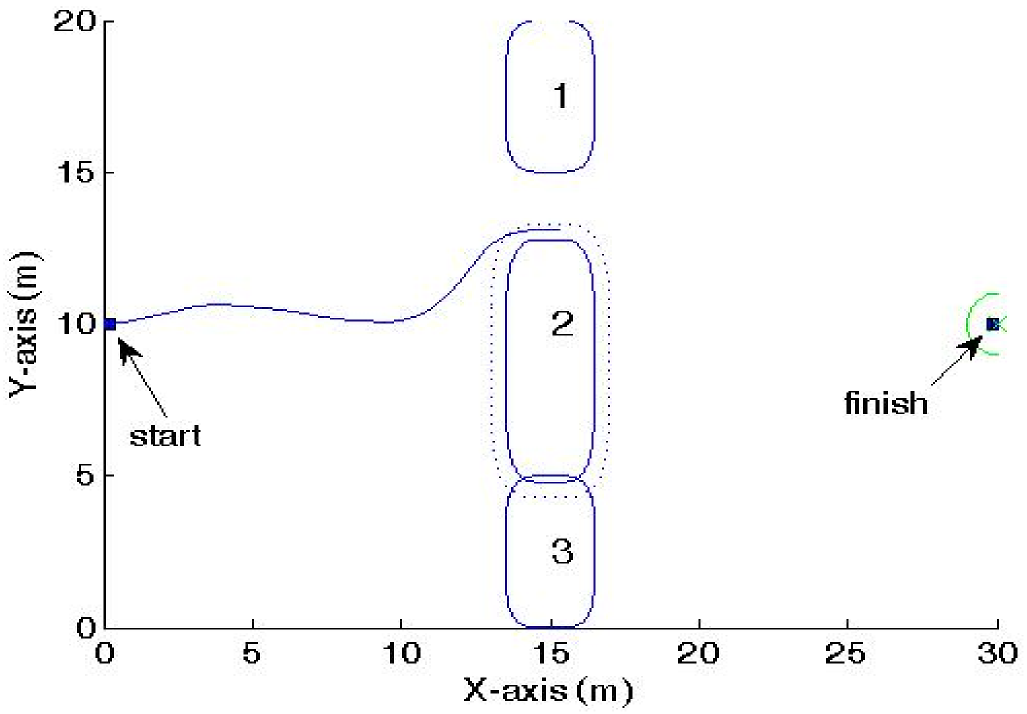


Figure 106. Cyclic Sliding Door (Prediction // *0.6 m/s* cycle speed).

Finally, the cyclic sliding door scenario was repeated for varying cycle speeds, but this time more information was known by the algorithm; specifically, the algorithm was given complete *a priori* knowledge of the motion of obstacle 2. Not only were the course and speeds known, but all the course and speed changes were also known in advance. Knowing the exact future position of obstacle 2 resulted in a trajectory that headed directly for the correct position to give the vehicle safe passage. No replanning was necessary. Maneuver times at all cycle speeds were the fastest possible (*33.4 sec*). Figure 107 shows the trajectory for a cycle speed of *0.6 m/s*, above which there can be no solution to the problem due to the physical limit on vehicle speed. It follows that with advance knowledge of environmental changes, the algorithm will generate feasible and safe trajectories up to the physical limits of the vehicle.

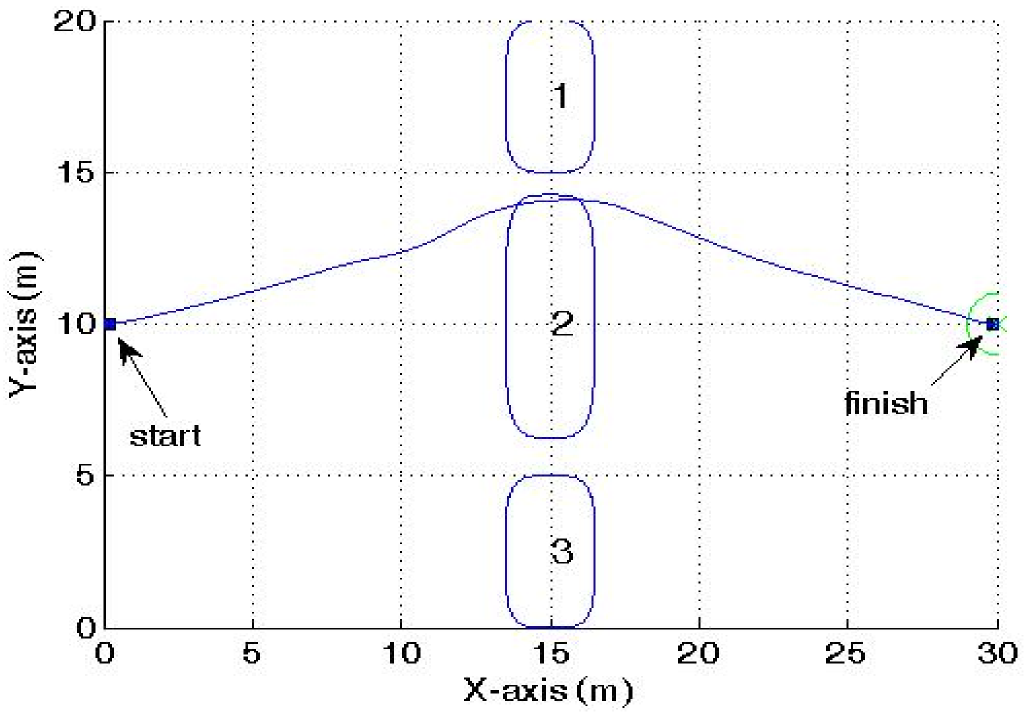


Figure 107. Cyclic Sliding Door (*a priori* Knowledge // *0.6 m/s* cycle speed).

Table 5 shows a summary of the results of running the cyclic sliding door scenario at varying cycle speeds up to the vehicle's limit using the three levels of information known to the algorithm. As expected, being able to predict obstacle positions resulted in a higher success rate and more near time optimal trajectories than using snapshots; and having *a priori* knowledge of environmental conditions produced even better results. However, knowing the future positions of obstacles beyond predicting them from their current course and speed is highly unlikely. It is assumed in this work that if an obstacle's course and speed changes can be known in advance, then it is likely another vehicle being controlled by the same algorithm or in direct communication with the vehicle whose trajectory is being planned. Either way, it would fall under multi-vehicle trajectory planning. These ideas will be analyzed later in this work. For this reason, further motion studies will be limited to the comparisons between using prediction based on course and speed and not using prediction (snapshots).

Table 5. Summary of Results for Cyclic Sliding Door Scenario.

Cycle Speed (*m/s*)	Maneuver Time (*sec*) Snapshots	Maneuver Time (*sec*) Prediction	Maneuver Time (*sec*) *a priori* Knowledge
0.1	33.5	33.4	33.4
0.2	collision	40	33.4
0.5	collision	36.3	33.4
0.6	collision	collision	33.4

3. Obstacle Crossing (No Intercept)

This scenario can be described by any obstacle whose motion results in it crossing over the trajectory of the vehicle, but does not pose a danger of collision. Figure 108 shows an example of the expected effect of a crossing obstacle on the vehicle's trajectory. The obstacle starts out motionless, but after *5 sec* it moves (from *x,y = 20,5* to *x,y = 20,25* at *1.5 m/s*) across the vehicle's path as the vehicle travels from *x,y = 0,15* to *x,y = 30,15*. The result shown in Figure 108 was generated using course and speed to predict obstacle position. The overall maneuver time was *32.3 sec*. If prediction is not used for a crossing obstacle scenario, the results are significantly worse. The scenario of Figure 108 was repeated using snapshots, and the resulting trajectory is shown in Figure 109a with a maneuver time that is significantly higher at *49.9 sec*. Figure 109b shows the trajectory's evolution over time; it shows that due to the sensitivity of numerical techniques (DIDO in this case) to the bias that is given, the trajectory is pushed over as the obstacle crosses it, even though no collision course existed. The trajectory essentially gets shaped by any obstacles that cross its path. The same effect can be produced by laying a string on a table whose endpoints correspond to the desired start and goal positions, then sliding objects around the table across the string, resulting in the string wrapping itself around those objects. Adding prediction allows obstacles to skip over the string (trajectory) when encountered. Another instance that would allow for the obstacle to cross without adversely affecting the trajectory is in the case that the obstacle travels fast enough to cause it to skip over the trajectory in successive environment snapshots.

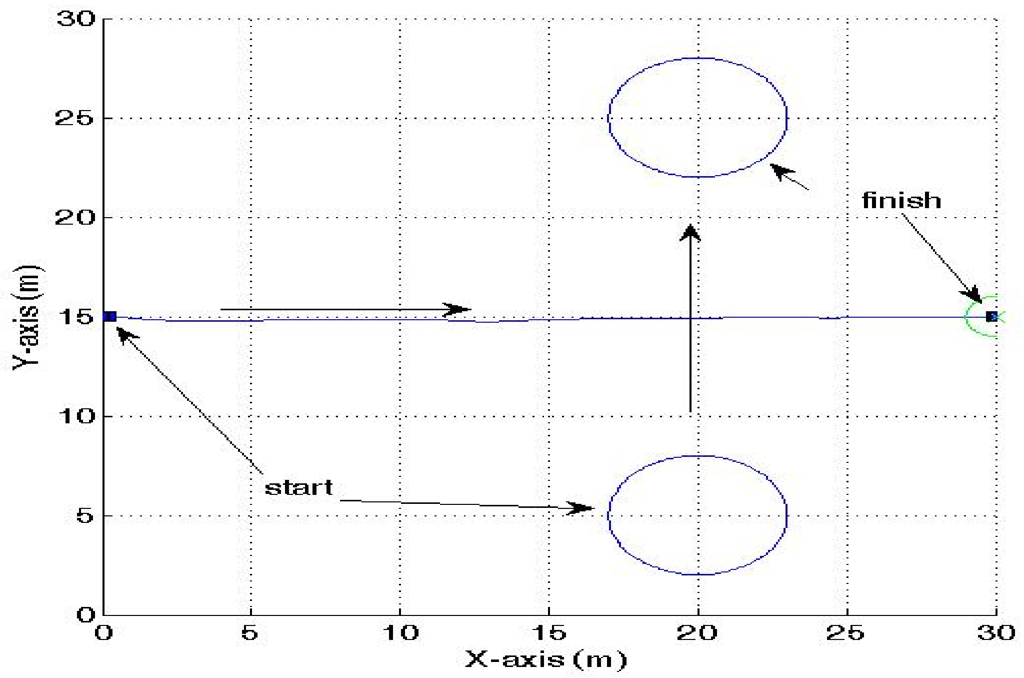


Figure 108. Obstacle Crossing Vehicle Path (Prediction).

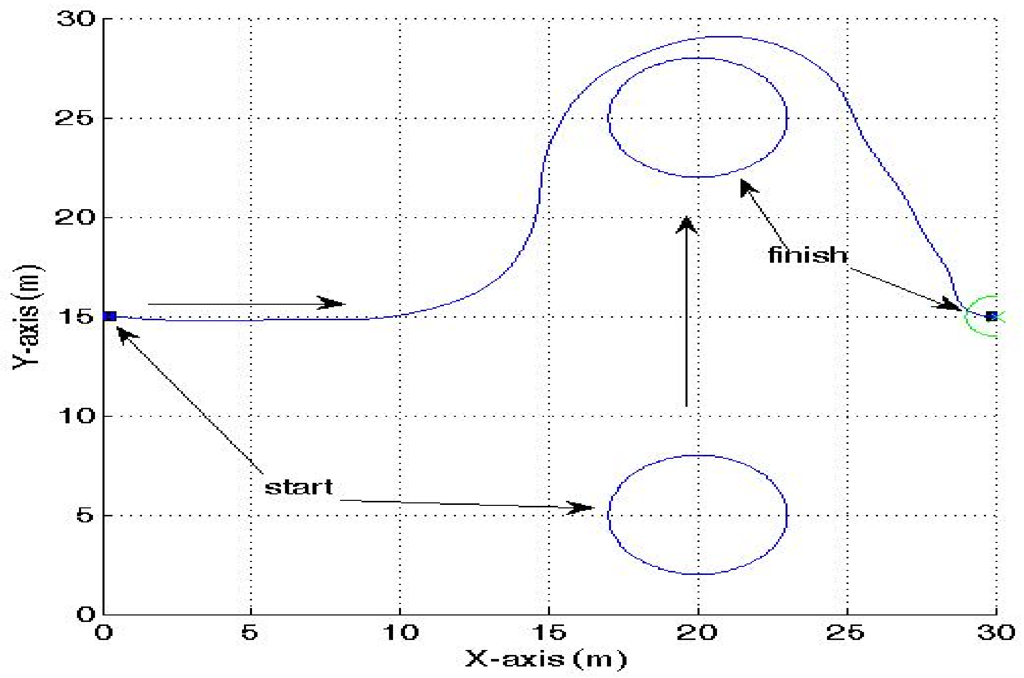


Figure 109a. Obstacle Crossing Vehicle Path (Snapshots).

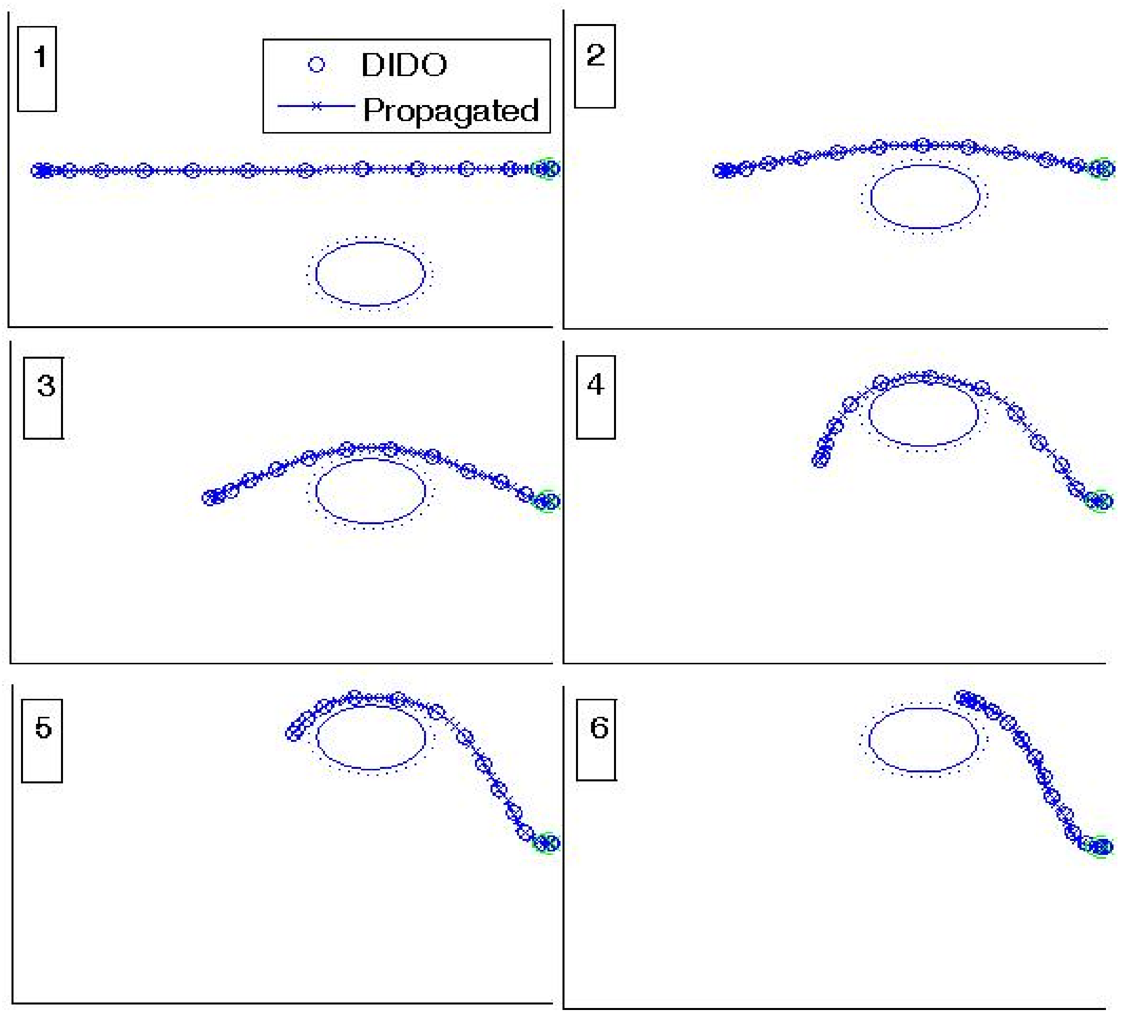


Figure 109b. Evolution of Obstacle Crossing Scenario (Snapshots).

4. Obstacle Intercept

The same obstacle crossing scenario was used, except the obstacle was slowed down to *0.6 m/s*, which put it on a collision course with the vehicle. Figure 110a shows the trajectory of the vehicle when using prediction, and the maneuver time was *34.5 sec*. The time evolution of the trajectory is shown in Figure 110b. For the first *5 sec*, before obstacle motion begins, the vehicle heads straight at the goal. Once the vehicle detects obstacle motion, it estimates the obstacle's course and speed, and autonomously makes a course correction to avoid collision.

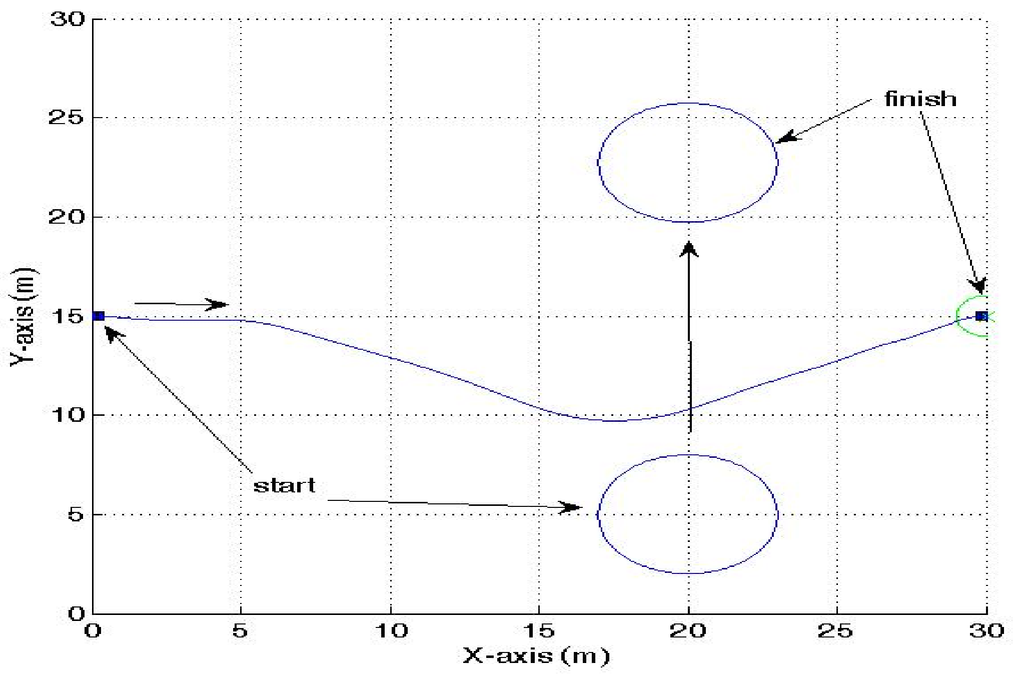


Figure 110a. Obstacle Intercept (Prediction).

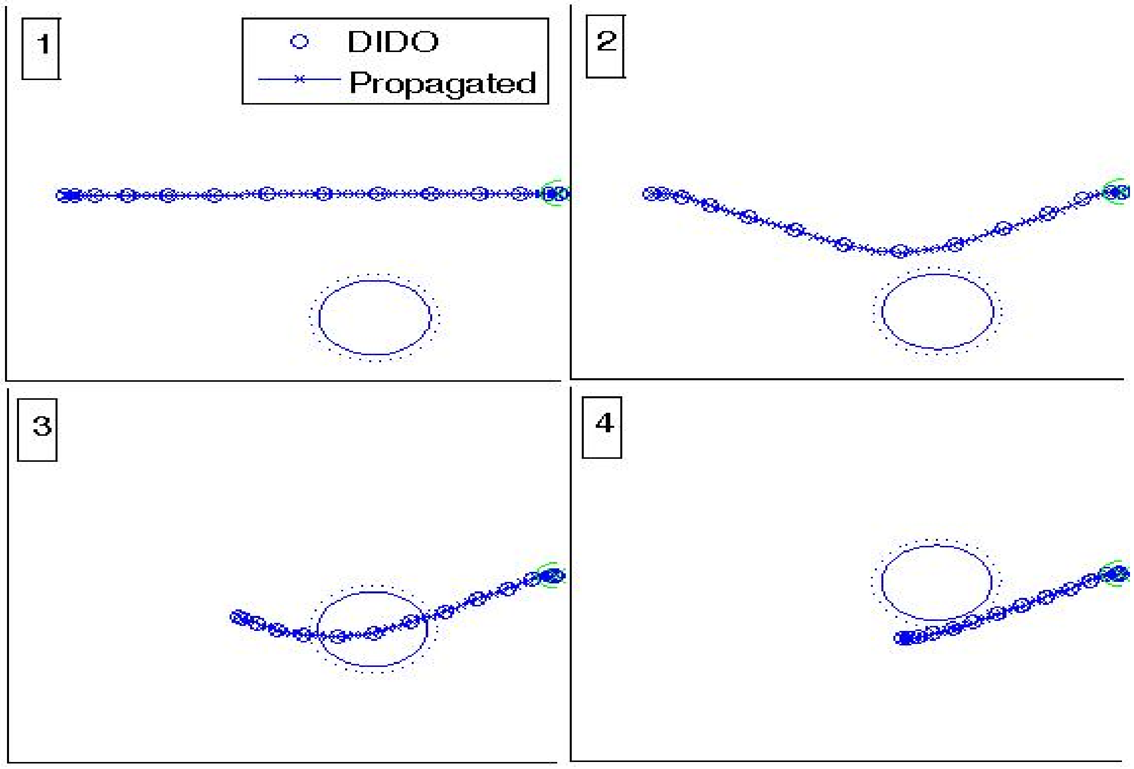


Figure 110b. Evolution of Obstacle Intercept (Prediction).

The obstacle intercept scenario was repeated using snapshots. The resulting trajectory collided with the obstacle, as shown in Figure 111. This again shows the weakness of using snapshots in a dynamic environment, because collisions cannot be avoided when an obstacle in very close proximity moves unpredictably.

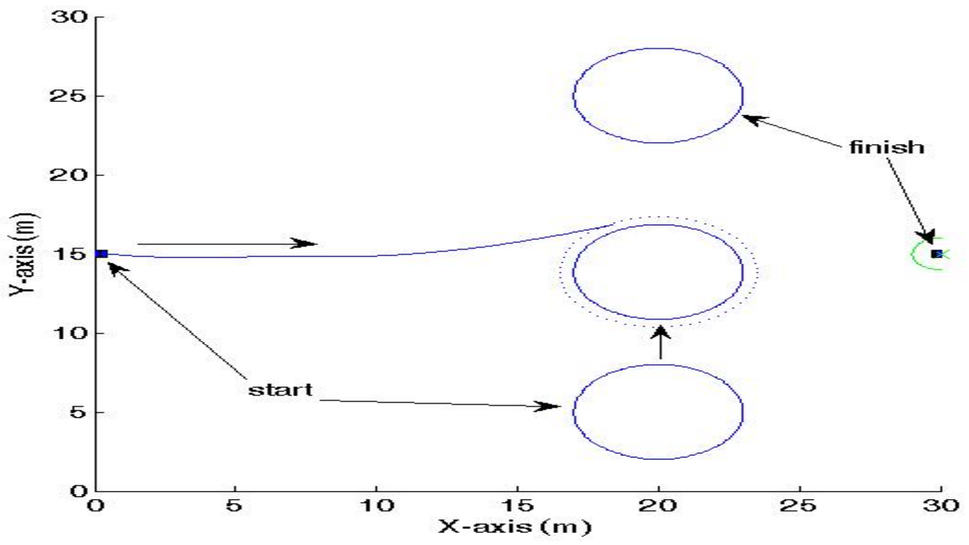


Figure 111. Obstacle Intercept (Snapshots).

5. Obstacle Intercept Window

The intercept window is defined in this work as a finite period in an obstacle's motion in which it is on a collision course with the vehicle, but the obstacle's course and speed is not constant, therefore the threat of collision only exists during a finite period of time. For example, two cars approaching an intersection have an intercept window, but then one car turns onto another road or slows down, thus eliminating the chance of intercept. The obstacle intercept problem was repeated, but the obstacle's path was modified to stop at *(x,y) = (20,10)*, which was prior to any collision. In this scenario the obstacle never obstructs the vehicle's path to the goal. Figure 112 shows the vehicle trajectory when snapshots are used. The small alteration in the vehicle's trajectory is due to the effect of the robustness on the cost of the maneuver. The overall maneuver time was *32.3 sec*.

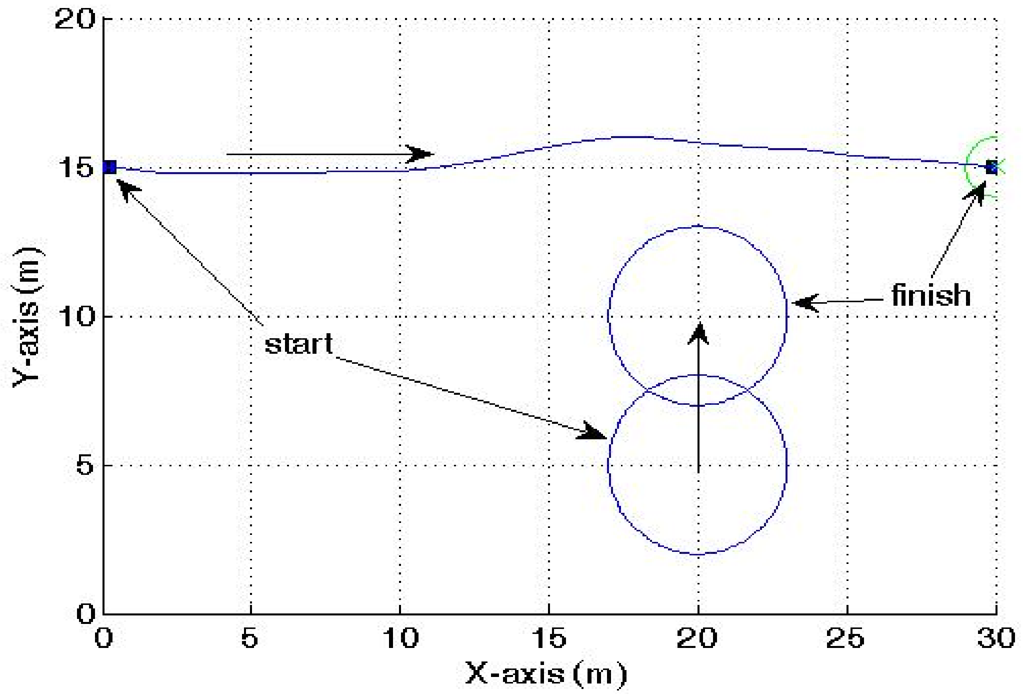


Figure 112. Obstacle Intercept Window (Snapshots).

The scenario was repeated using prediction, and the resulting trajectory is shown in Figure 113a. The vehicle was baited off its original trajectory due to the fact that it incorrectly predicted the obstacle's collision course and autonomously replanned to avoid the collision. The maneuver time for this scenario was *33.8 sec*, which is longer than when snapshots were utilized. The evolution of the trajectory is shown in Figure 113b. Frame 1 shows the trajectory before obstacle motion commenced. Frames 2 and 3 show the predicted trajectory once the algorithm determined the obstacle's course and speed. Frame 4 shows the new trajectory after the obstacle stopped. Essentially, the obstacle's motion faked out the vehicle into executing an extra maneuver, resulting in it taking a longer time to reach the goal. This intercept window scenario is the one instance when using snapshots generates a faster maneuver than when using prediction. However, given the fact that prediction still produced a valid trajectory (albeit a little longer) and the fact that in all other cases prediction is more effective, this issue is an acceptable disadvantage.

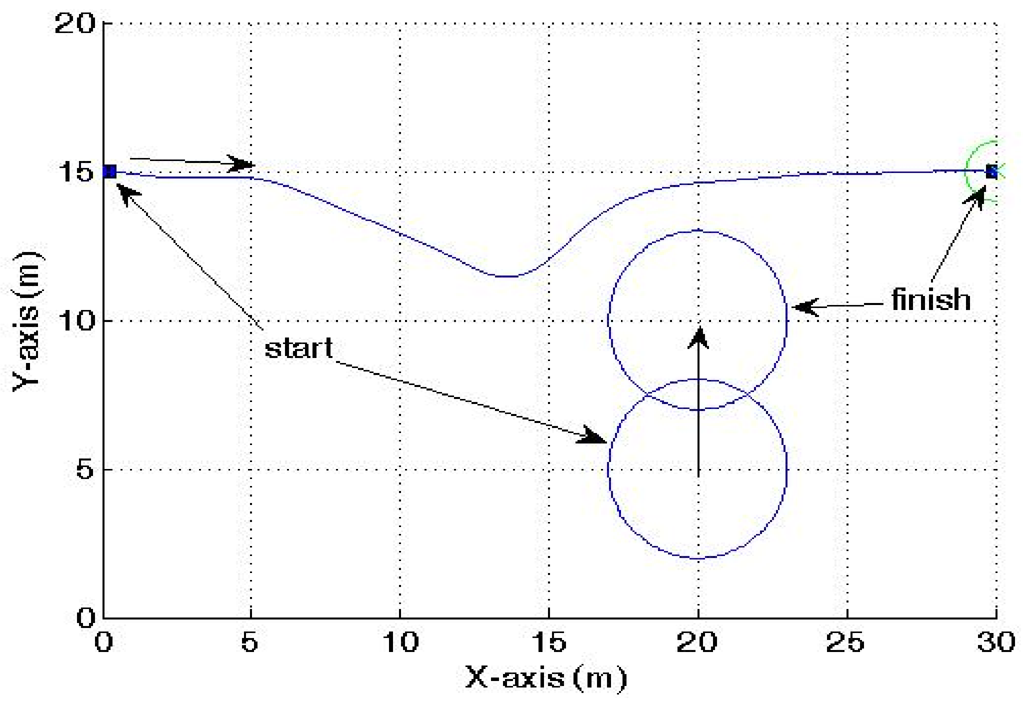


Figure 113a. Obstacle Intercept Window (Prediction).

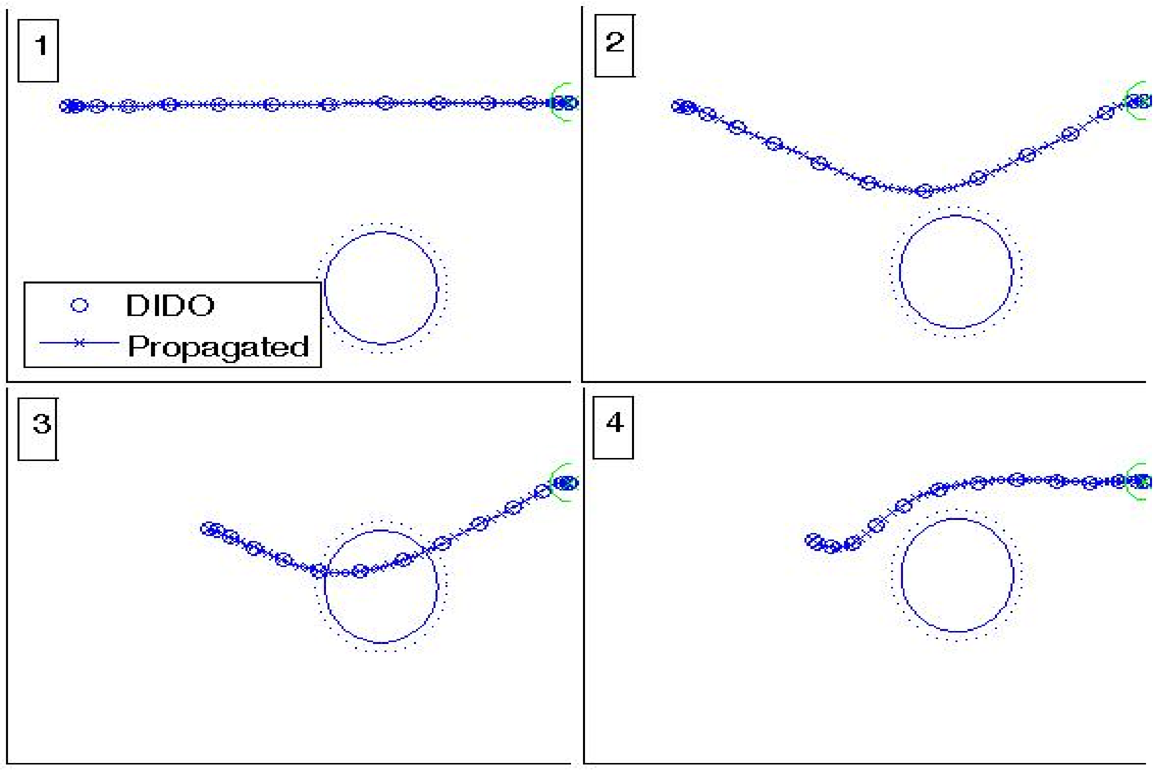


Figure 113b. Evolution of Intercept Window Scenario (Prediction).

6. An Improved Prediction Method

In the case of the obstacle intercept window discussed above, the act of predicting an obstacle's position using its instantaneous course and speed can have the effect of baiting the vehicle off its optimal trajectory. The scenario studied had only one moving obstacle. What if there were many moving obstacles or vehicles that caused the planned trajectory to change erratically? The solution to this situation is to modify the prediction method to only consider those moving obstacles/vehicles that are an imminent threat. This is done by modifying the algorithm when planning trajectories using prediction. A moving obstacle is only considered in the trajectory planning process if its propagated path intercepts the vehicle's caution zone (centered around its start position, (x_i, y_i), for that iteration) within the first eight seconds of the obstacle's trajectory $(\underline{x_o}, \underline{y_o})$, as determined by the following equation:

$$DR = \ln\left[\left(\frac{x_i - x_o}{8.5}\right)^2 + \left(\frac{y_i - y_o}{8.5}\right)^2\right] \tag{58}$$

If any element of $(\underline{x_o}, \underline{y_o})$ results in *DR<0* in Equation (58), then the moving obstacle comes close enough to include it in the trajectory planning formulation. The *8.5* in the denominator of Equation (58) accounts for the maximum caution zone that the vehicle can possess (*8m*), plus *0.5m* to account for vehicle size. Using eight seconds of the obstacle's trajectory establishes its caution zone, because it is based on twenty time steps at the maximum allowed run time of *0.4 sec*. If an intercept of the obstacle's first eight seconds of trajectory is not detected ($DR \geq 0$) with the vehicle's caution zone (based on its start position for that iteration), then the obstacle is treated as though it does not exist. This has the effect of allowing the vehicle to plan optimal trajectories without being affected by the movement of obstacles/vehicles that are too distant to consider, since their movements could likely change several times as they get closer. To illustrate, the intercept window scenario was repeated using the new prediction process, and the resulting trajectory is shown in Figure 114. The deviation in the vehicle's path is due to a combination of two issues. First, the vehicle senses the obstacle's trajectory within its caution zone prior to the obstacle stopping. Second, the effect of robustness on maneuver cost, as in Figure 112. This method of performing prediction won't prevent close obstacles from baiting the vehicle off its path, but distant obstacles will no longer affect the vehicle's motion. For the remainder of this work, this improved method will be utilized whenever using prediction to conduct trajectory planning.

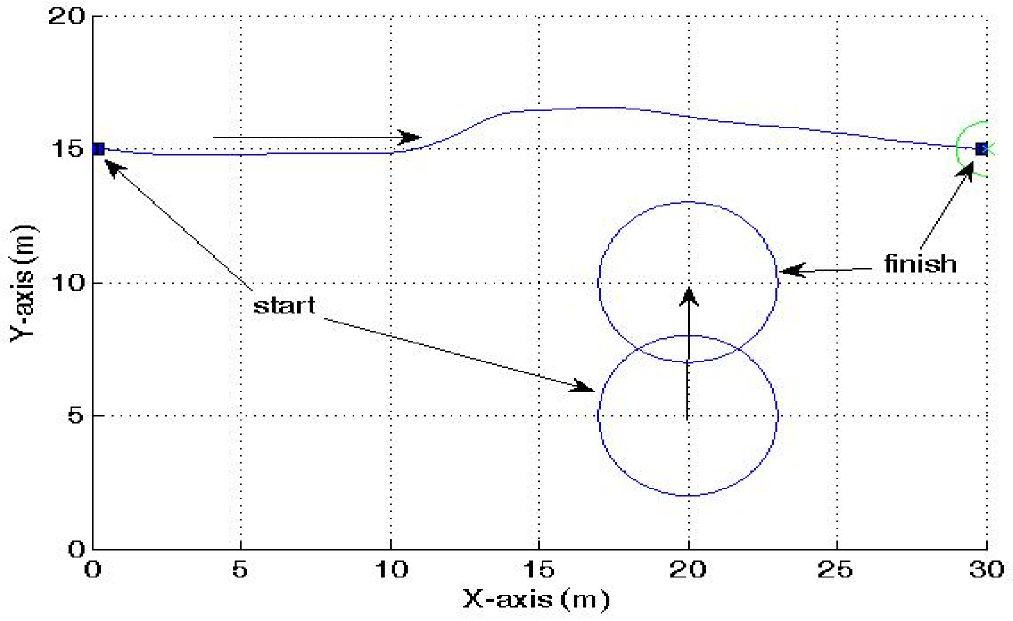


Figure 114. Obstacle Intercept Window (Improved Prediction).

C. TARGET MOTION STUDIES

1. Target Rendezvous: Vehicle Faster than Target

In this scenario the vehicle must rendezvous with a moving target. Figure 115a shows the vehicle trajectory with no target position prediction (snapshots), and Figure 115b shows the evolution of that trajectory. Initially, the target is motionless, and after *15 sec* it commences its movement at *0.6 m/s*. Since the algorithm does not use the target's course and speed to predict a rendezvous point, each trajectory planning iteration simply aims the vehicle at the current target position. This results in the vehicle falling in behind the target and catching up to it. The maneuver time was *37.2 sec*. Using the target's course and speed to predict a rendezvous point resulted in the trajectory of Figure

116a and a faster maneuver time of *33.3 sec*. Figure 116b shows the evolution of that trajectory; notice the trajectory predicts the target's future position and follows an intercept course.

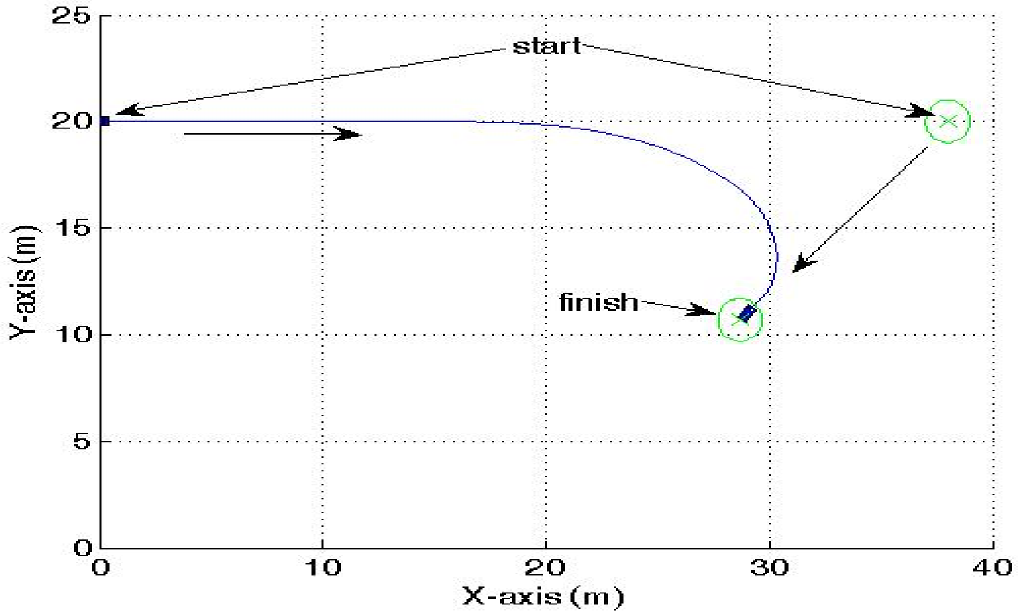


Figure 115a. Vehicle Faster than Target (Snapshots).

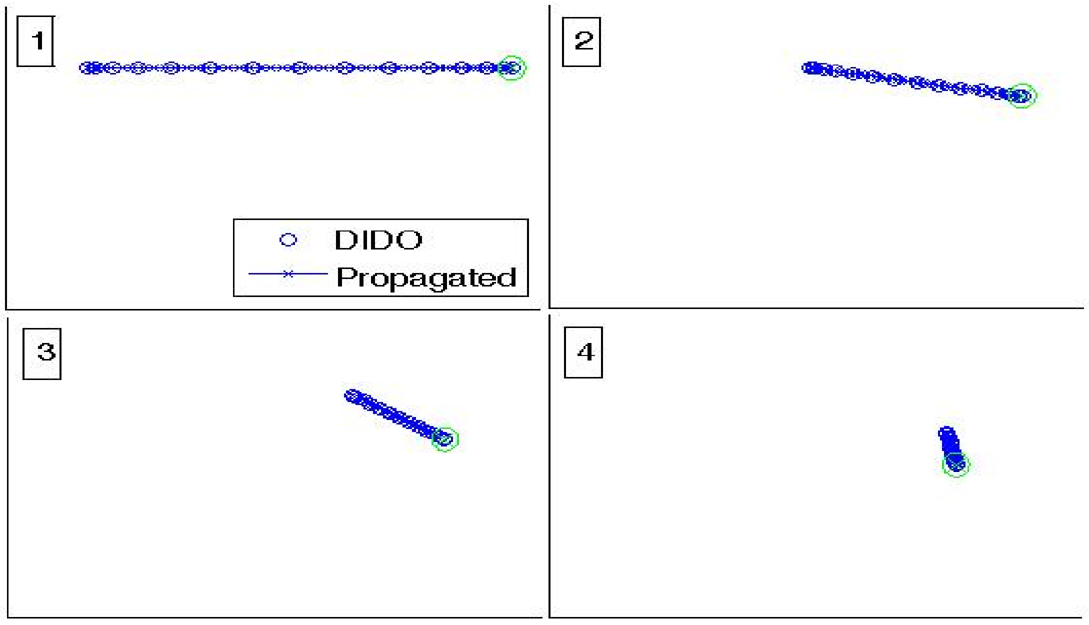


Figure 115b. Evolution of Figure 115a Trajectory.

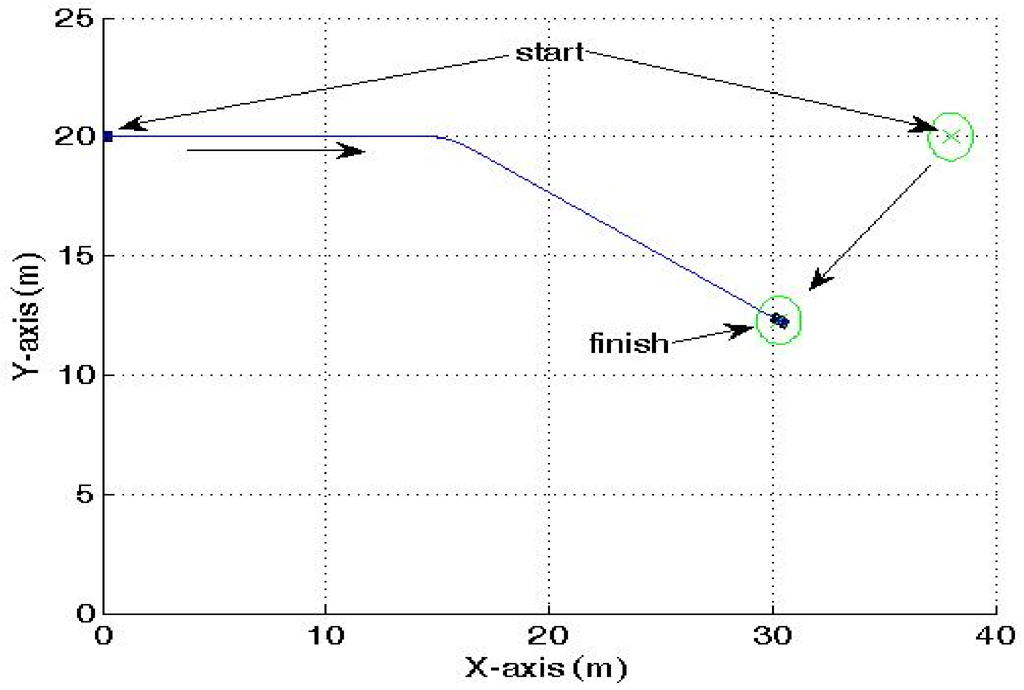


Figure 116a. Vehicle Faster than Target (Prediction).

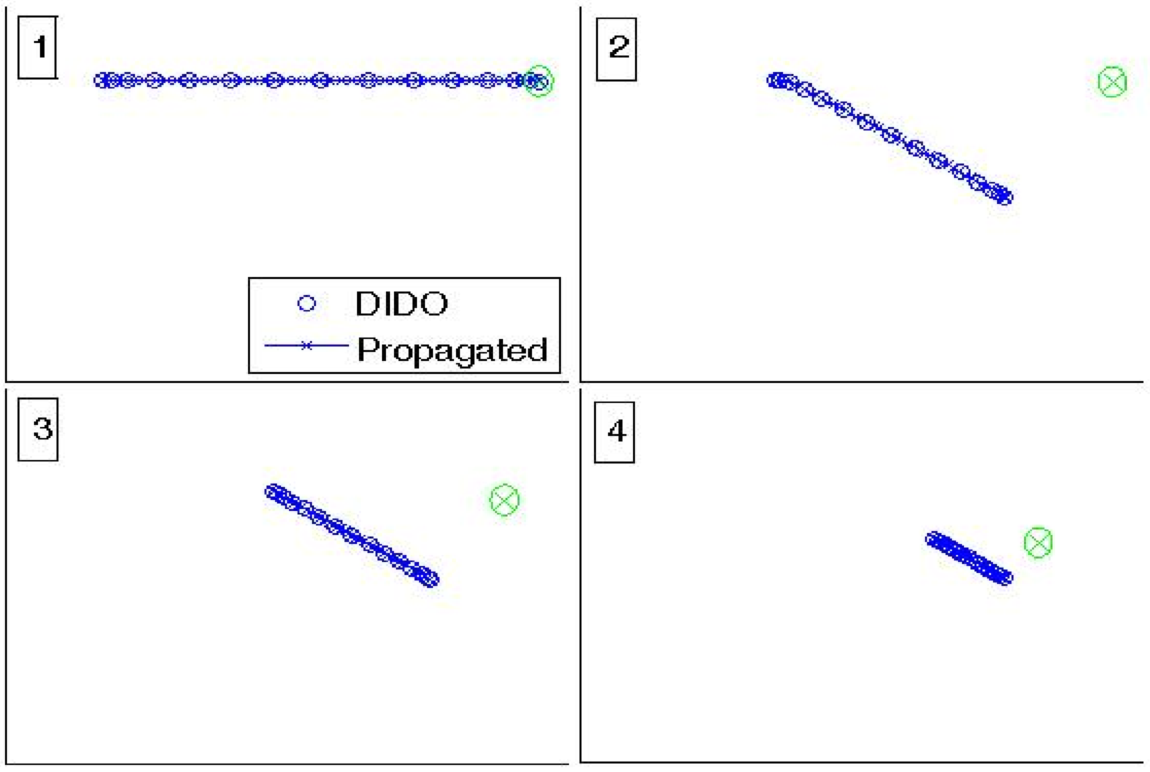


Figure 116b. Evolution of Figure 116a Trajectory.

2. Target Rendezvous: Vehicle Slower than Target

The same target rendezvous scenario was run, but this time target speed was faster at *1.2 m/s*. The resulting trajectory for the case of snapshots is shown in Figure 117. The vehicle falls in behind the target, but can never catch up. Unless the target slows down, the vehicle will never reach it. Figure 118 shows the trajectory if an intercept can be calculated using target course and speed to predict its future position. In this case, the vehicle was able to rendezvous with the target. It follows that if we want to maximize the vehicle's chances of being able to catch a moving target, then prediction is the way to achieve the best results.

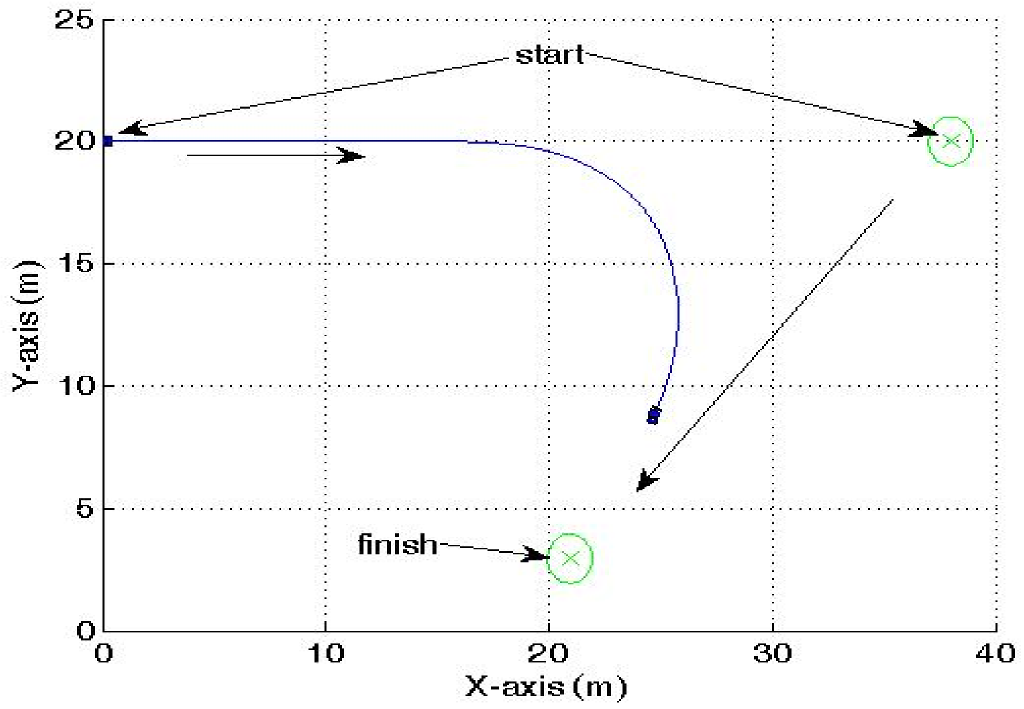


Figure 117. Vehicle Slower than Target (Snapshots).

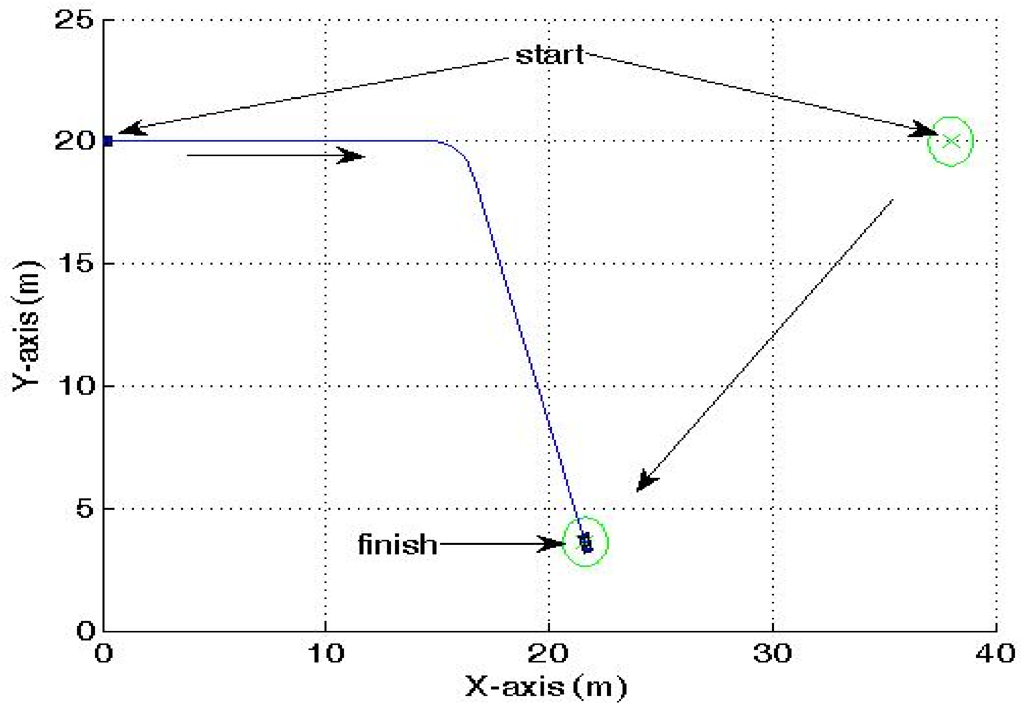


Figure 118. Vehicle Slower than Target (Prediction).

3. Target Rendezvous: Variable Target Motion

This section studies what happens if the target's motion changes during the vehicle's maneuver. The scenario is the same as that used for the *vehicle slower than target* case, except instead of the target moving indefinitely, it stops after *10 sec* of motion. Figure 119 shows both cases. The snapshots case produced a less tortuous maneuver as well as a faster one, posting a *33.9 sec* maneuver time as opposed to the *36.8 sec* maneuver time when prediction was used. This scenario shows that in a small number of cases, snapshots can actually produce better results than when using prediction. However, as stated previously, the price paid is acceptable when weighing the benefits of using prediction.

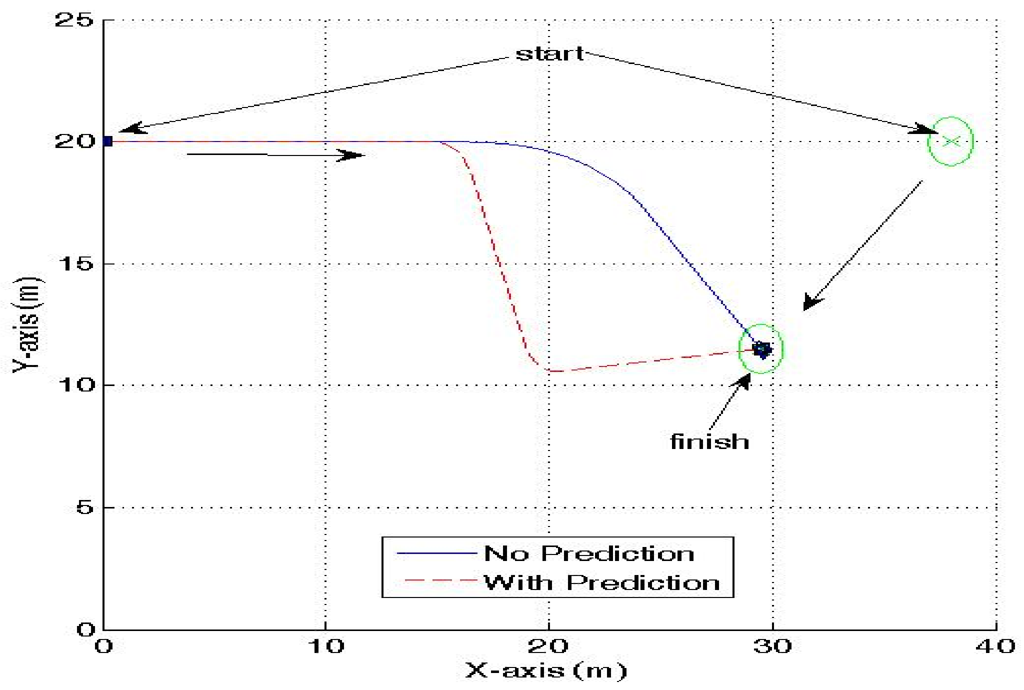


Figure 119. Variable Target Motion.

D. CONCLUSIONS

The simulations and discussions in this section show that the trajectory planning algorithm developed in Chapter IV is an information centric planner that produces better results with more information. Snapshots cannot predict possible collisions with moving obstacles, and thus collisions are more probable when in close proximity to those obstacles. Also, the trajectories are generally slower when using snapshots. Predicting obstacle and target positions using course and speed data produced faster trajectories (with a few exceptions) than using still snapshots of the environment. Having *a priori* knowledge of environmental conditions will always produce the best results. It should also be noted here, in keeping with the goals of this work, that all closed loop run times for all simulations conducted in this chapter were well within the requirement of keeping below *50 sec*. Also, no degradation in any of the performance criteria of Chapter II was suffered in the performance of simulations thus far.

VI. PORTABILITY STUDIES

A. BACKGROUND

The portability of the optimal control approach has already been demonstrated in numerous ways in this work and others. With the proper problem formulation, it has been shown that this technique can be applied to various vehicle and environment types while optimizing a collection of chosen parameters. Optimal control trajectory planning has been applied to a tricycle [1, 52], a UGV [1, 51, 52, 54], a UAV [1, 51, 52, 53, 54], an inverted pendulum problem [1, 52], the NPSAT1 Spacecraft [49, 51, 55], an RLV [50, 51], and the International Space Station that included two successful flight implementations.

In the *Path Planning and Control* performance parameter, it has been stated that all the other path planning methods require the addition of a trajectory following subsystem. This is because the path to the goal is determined, but not the controls. The optimal control technique solves the control solution as well; therefore, a path follower is not required. However, the optimal control technique can be used as a path follower if desired. What if the desired path is already known? Solving the trajectory in that case would not be necessary, and all that would be needed is a path follower. The high portability/flexibility of the optimal control technique allows for its use as a path follower. This is a strong utility, because it can be used to designate preferred paths, such as a route that may not be the time optimal path, but is safer for the vehicle. The path following capability of the optimal control technique will be studied in this chapter.

B. PATH FOLLOWING FORMULATION

Converting the solution from that of a minimum time/maximum robustness problem to that of a path following problem involves one simple change to the problem formulation. Specifically, this change is made to the original cost function, which is repeated here as Equation (59):

$$J[\underline{x}(\cdot),\underline{u}(\cdot),t_f] = t_f + \int_0^{t_f} r(t)dt \tag{59}$$

A path following cost, *p(t),* is added to Equation (59). This path following cost essentially adds to the overall cost of the trajectory when the vehicle is off the desired path. The new formulation for the overall cost becomes Equation (60):

$$J[\underline{x}(\cdot),\underline{u}(\cdot),t_f] = t_f + \int_0^{t_f} \left(r(t) + p(t)\right) dt \tag{60}$$

The path cost is simply a function of the vehicle's deviation from the desired path. Recall that proper problem formulation requires all variables be well scaled and balanced, and all functions must be smooth. If a quadratic path cost is used, it would become very large as the deviation from the path increased, which would require one of two approaches to scale and balance it with the robustness cost. First, it could be given a weighting factor to scale it down, but then the weighting factor would have to vary as a function of path deviation; the higher the path deviation, the lower the weighting, in order to prevent the path cost from dominating the cost equation. Second, the quadratic cost could simply be given a saturation point (maximum value) equal to the maximum value of the robustness cost, but then the function would not be smooth. Either case is not favorable. The inverse tangent function was chosen over a simple quadratic because it is in keeping with proper problem formulation. It is shown here as Equation (61), where $\underline{x}$ is the actual vehicle path vector and $\underline{y}$ is the desired path vector:

$$p(t) = \tan^{-1}\left[\left(\underline{x} - \underline{y}\right)^2\right] \tag{61}$$

The inverse tangent function allows the path cost to be scaled down to the range of the robustness cost without the addition of a varying weight factor, thus balancing the two costs; and it does this scaling using a smooth function. The path cost, Equation (61), is plotted on Figure 120, which also shows a plot of the robustness cost from Figure 86 and a simple quadratic path cost.

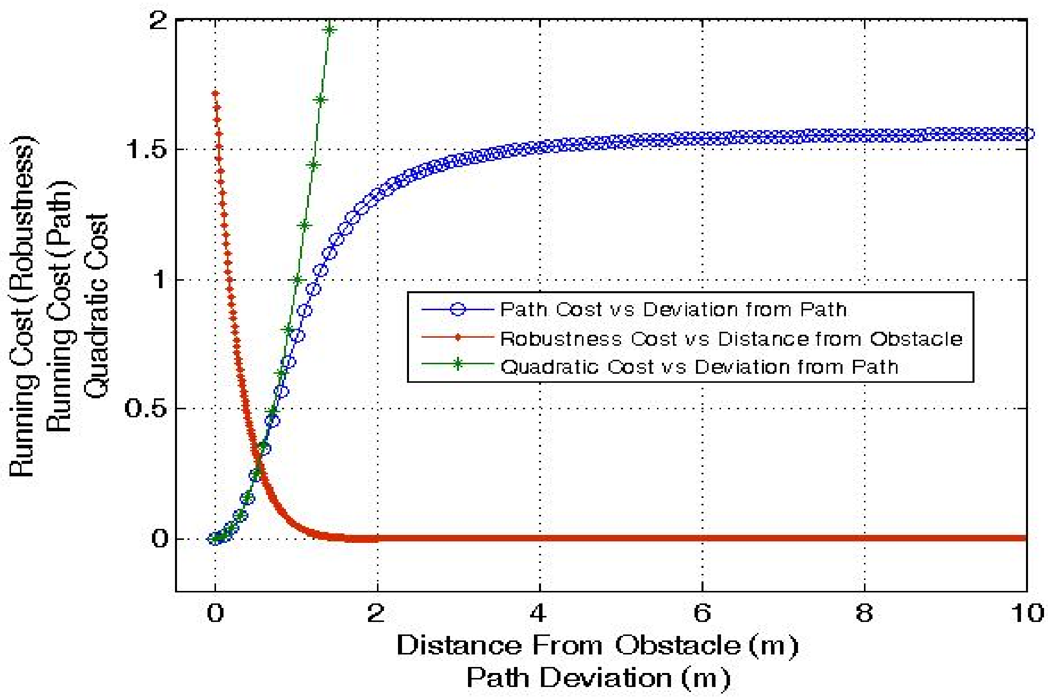


Figure 120. Running Cost Comparison (Robustness vs. Path Cost).

C. PREFERRED PATH // SAFETY

The first utility of path following to be discussed is the preferred path. Why might there be a preferred path other than the time optimal path? There are numerous reasons for a preferred path; the user may simply want to indicate a path that is less bumpy for the vehicle as it surveys the area; or perhaps the operator wants to steer the vehicle around a mine field; or maybe the user may want the vehicle to follow a pattern through a designated area, such as it would if it were conducting mine hunting operations. The reasons for path following are numerous. Figure 121 shows the simulation results of several path following scenarios. For each scenario, a different combination of start and finish points was chosen. In all cases, the vehicle was given the same desired path, which is shown on Figure 121 and given here as Equation set (62):

$$\begin{aligned} y &= 0 \ldots\ldots\ldots\ldots\ldots\ldots\ldots x < 5, x > 36.4 \\ y &= 10\sin\left(\frac{x-5}{10}\right) \ldots\ldots 5 \le x \le 36.4 \end{aligned} \qquad (62)$$

Figure 121 demonstrates that regardless of the start or finish points, the vehicle will seek out the path first, and then follow it as far as it can before leaving it again.

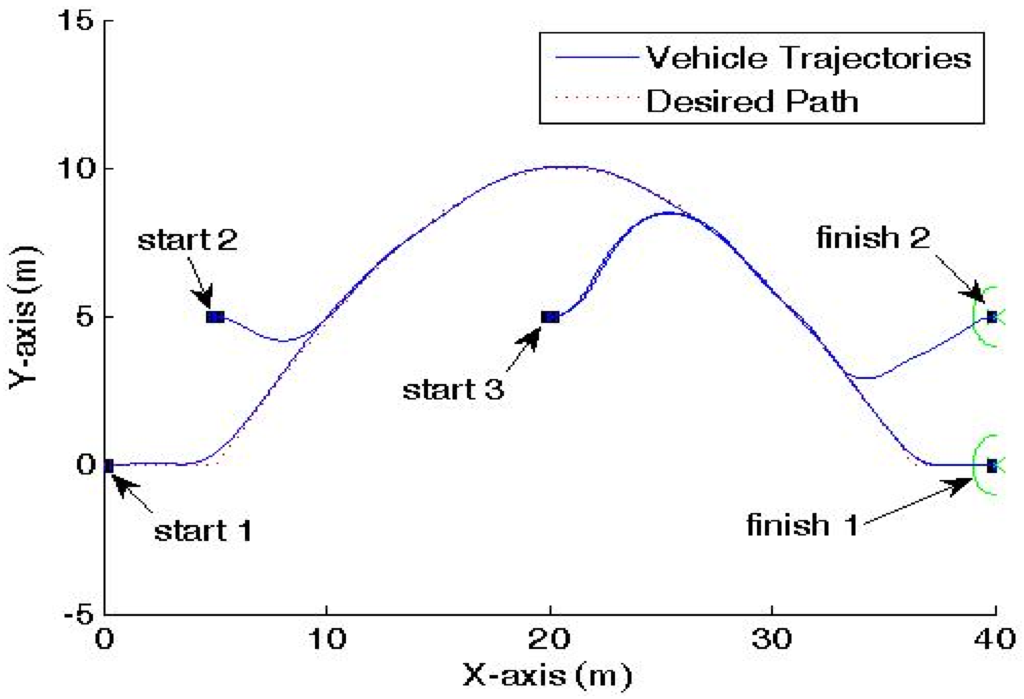


Figure 121. Preferred Path // Safety Path Following.

D. PATH FOLLOWING WITH OBSTACLE AVOIDANCE

What if a vehicle is conducting mine hunting operations which require it to follow a specific path in order to achieve complete coverage of an area? In this scenario, trajectory planning without the path following cost would fail to provide the necessary coverage, however, if path following is added to produce the cost of Equation (60), then complete coverage can be achieved. Two examples are shown here. Figure 122 compares the use/non-use of path following with trajectory planning in the case of a vehicle being required to stay on its designated path ($y=10$) with a known obstacle in its way. As is evident here, if path following is not included, the trajectory planning method never follows the desired path, because it simply seeks the time optimal path from start to finish. However, using path following results in the vehicle remaining on the path at all times, except to deviate from the path in order to avoid the obstacle. Figure 123 shows the trajectories of a vehicle hunting for a mine. When a mine is found, the trajectory planning method (without path following) treats it as an obstacle avoidance problem and does not reacquire the path after the obstacle. With path following included, the vehicle avoids the mine and reacquires its path to continue mine hunting.

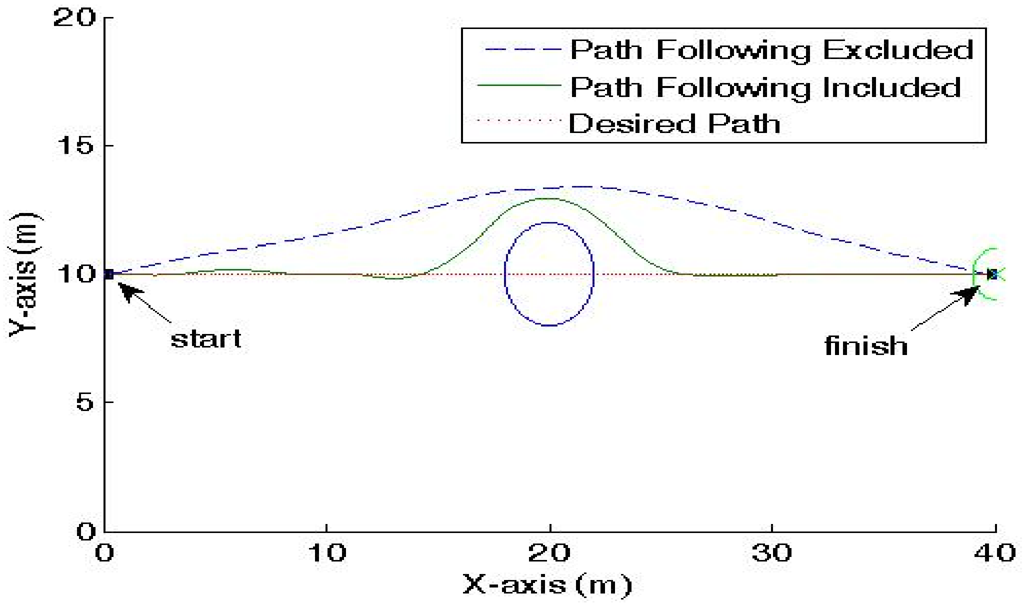


Figure 122. Path Following vs. Trajectory Planning with Known Obstacle.

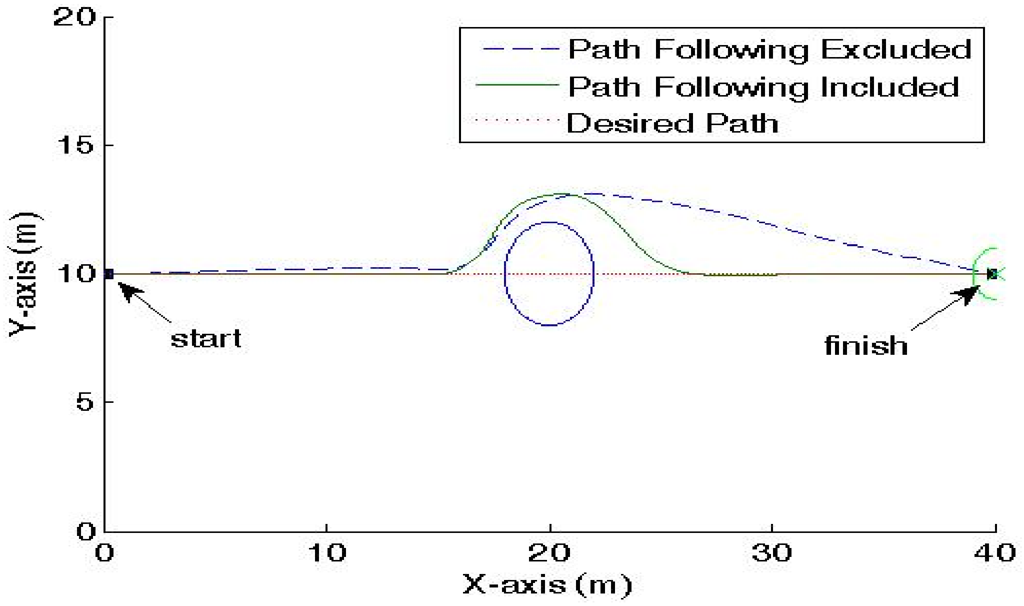


Figure 123. Path Following vs. Trajectory Planning with Pop-up Obstacle.

The problem of having to tune multiple running cost parameters at once is eliminated by performing a switching action on the running cost in Equation (60). This is achieved by using the path following portion of the running cost only, until the vehicle feels the effects of a nearby obstacle. The influence of nearby obstacles is measured using the robustness function. When the robustness function exceeds a threshold, the algorithm switches from path following to obstacle avoidance. Equation (63) describes this switching action:

$$\begin{aligned} J[\underline{x}(\cdot),\underline{u}(\cdot),t_f] &= t_f + \int_0^{t_f} p(t)dt........r(t) < threshold \\ J[\underline{x}(\cdot),\underline{u}(\cdot),t_f] &= t_f + \int_0^{t_f} r(t)dt........r(t) \geq threshold \end{aligned} \tag{63}$$

The next question is how to tune the path cost, *p(t)*, to the minimum time. The answer is that it is not necessary. When the path is already known, there is no need to plan a trajectory from start to finish. The trajectory can be broken up into segments, which would prevent the minimum time cost from dominating over the path cost. The other way to conduct path following is to use a leader-follower method. In this method the vehicle shoots for a point on the path that moves with the vehicle. Both of the above methods are forms of receding horizon planning discussed in [62]. Figure 124a shows the trajectory of a path following/obstacle avoidance problem using the leader-follower technique as discussed above. The desired path is defined as *y=40* on the first segment, and then *x=44* on the second segment. The vehicle is continuously planning a trajectory to a point that is on the path and at the outer edge of the vehicle caution zone (i.e., the endpoint is *8m* away from the vehicle along the desired path). Figure 124b presents the evolution of the leader-follower problem. Frames 1, 2 and 3 show the vehicle switching to obstacle avoidance. Frame 4 shows the trajectory of the vehicle as it rounds the corner of the desired path. The trajectory shown in each frame looks like a train moving along its track, but with the added ability to leave its track for obstacle avoidance.

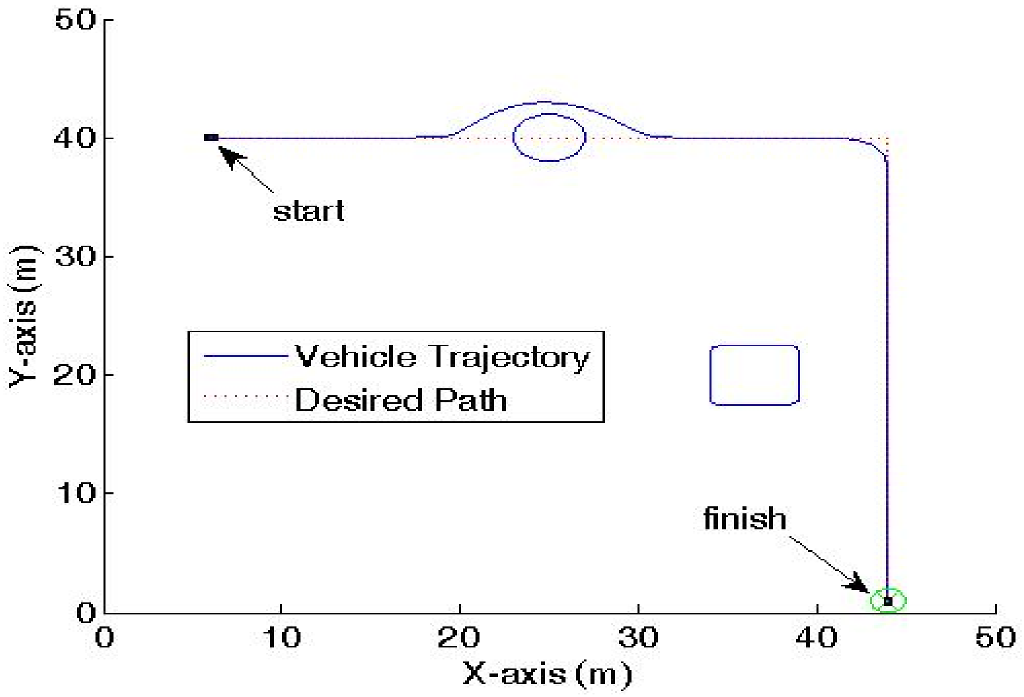


Figure 124a. Leader-follower Trajectory.

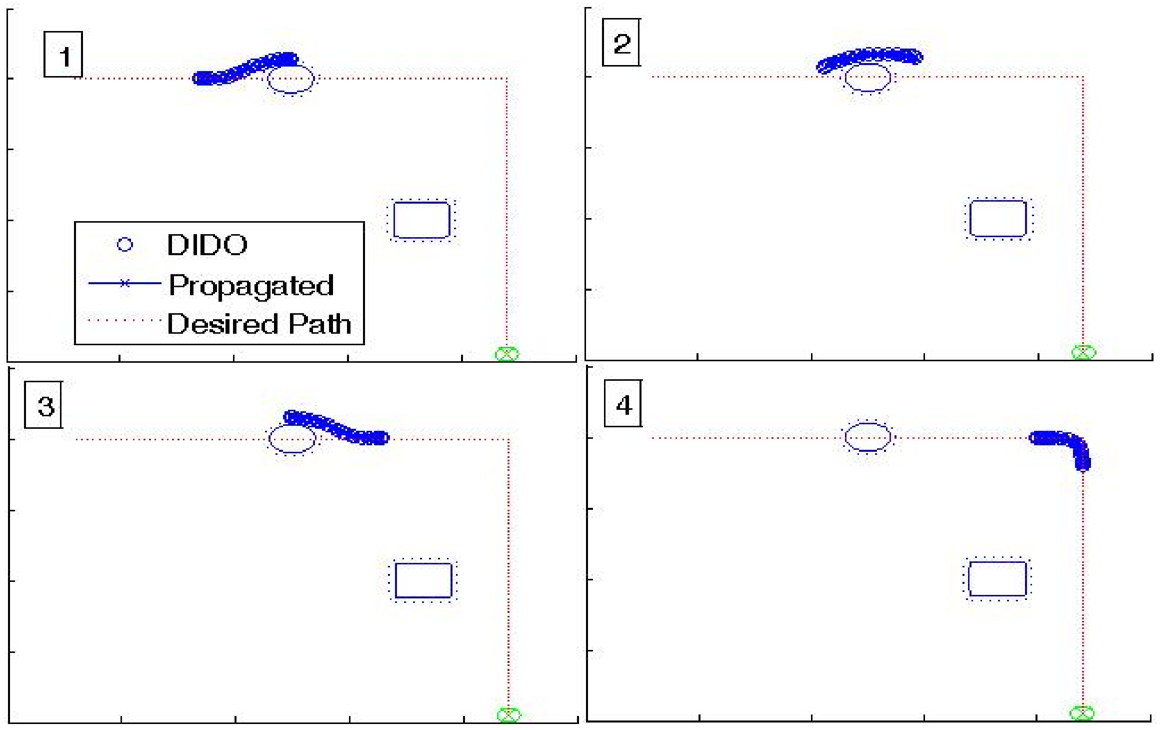


Figure 124b. Evolution of Leader-follower Scenario.

E. NARROW PASSAGE PROBLEM RE-VISITED

Using the path following concept can solve such things as the narrow passage problem of Chapter IV.D.3. Figure 125a shows a problem solved by the trajectory planning algorithm involving the avoidance of one obstacle and the adherence to a preferred path (*y=10*), shown as a red dotted line in the figure. In reality, the environment could have consisted of a narrow passage between two obstacles (Figure 125b) or a safe corridor through a mine field (Figure 125c). The use of path following allowed for the solution of those problems using just the environment shown in Figure 125a, without having to add more path constraints, and without having to follow the narrow passage rules developed earlier which would have involved raising the number of nodes being used per iteration.

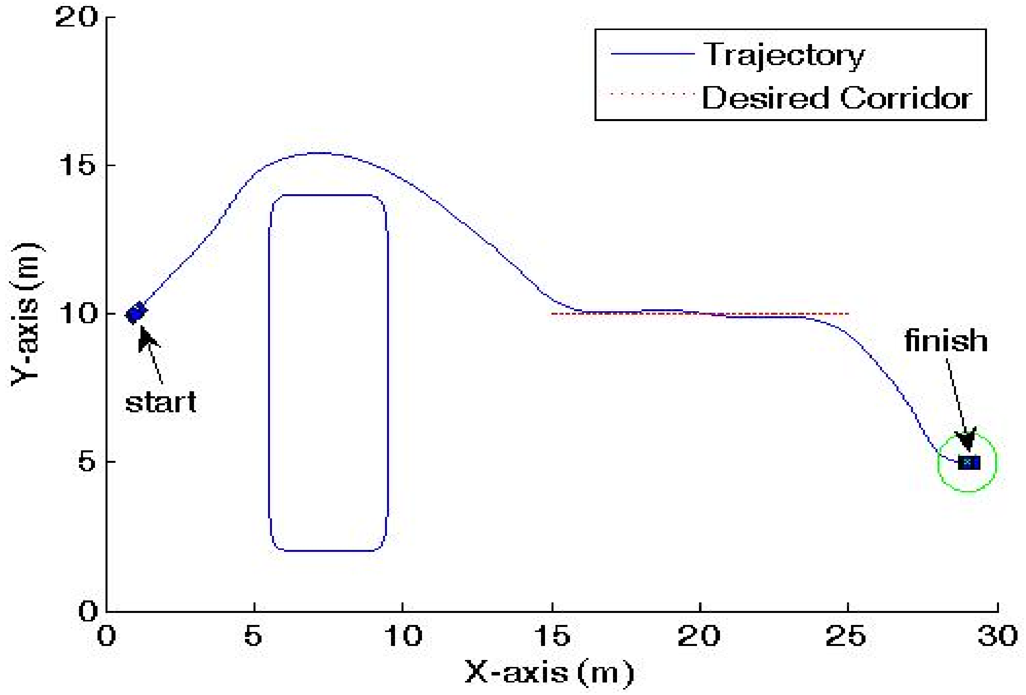


Figure 125a. Trajectory Planning Problem with Preferred Path.

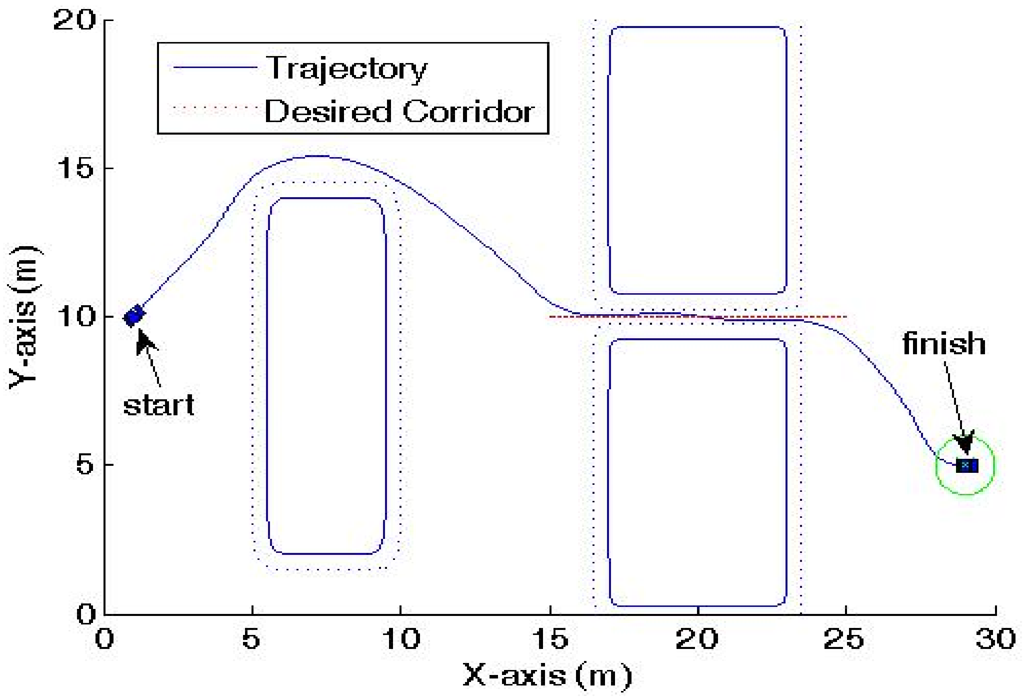


Figure 125b. Narrow Passage between Two Obstacles.

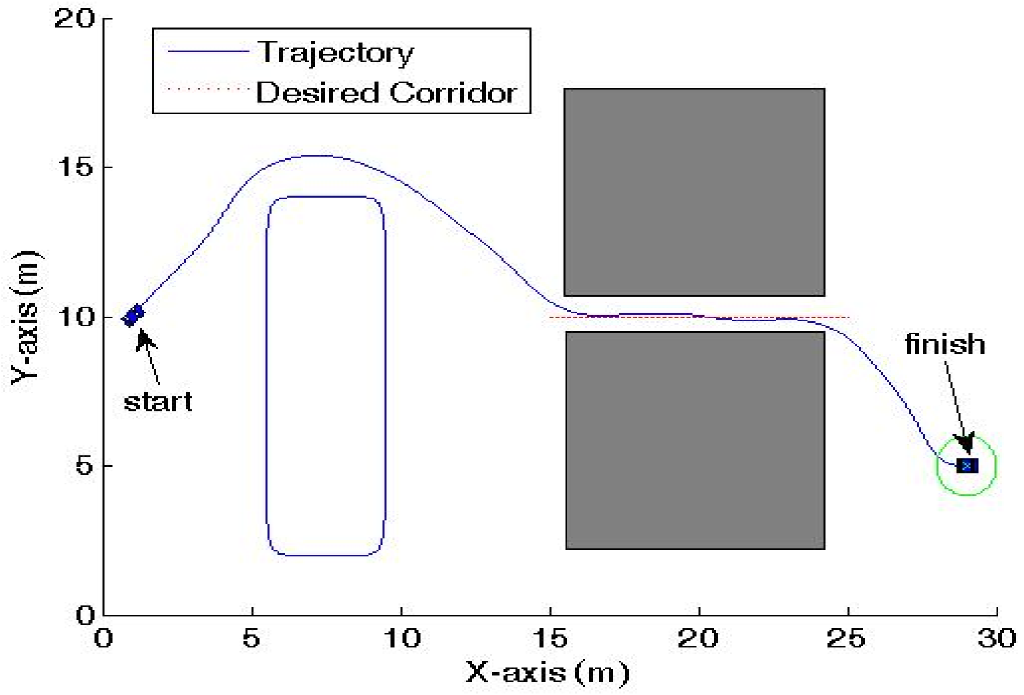


Figure 125c. Preferred Corridor through a Mine Field.

F. CONCLUSIONS

This chapter has shown the path following capability of the optimal control technique. It has demonstrated the flexibility of the technique through its use as a path follower and the ability to use it as a combination trajectory planner and path follower. Prior to this chapter, the technique was used to optimize time and robustness. This chapter has shown that it can be used to optimize other things, such as minimizing a vehicle's deviation from a path. There seems to be no limit to the myriad of problems that can be solved using optimal control techniques; the main issue here then becomes the proper way to formulate the problem.

In keeping with the goals of this work, all closed loop run times for all simulations conducted in this chapter were well within the requirement of keeping below *50 sec*. Also, no degradation in any of the performance criteria of Chapter II was suffered in the performance of simulations thus far.

VII. MULTI-VEHICLE TRAJECTORY PLANNING

A. BACKGROUND

Future robotic missions will call for the autonomous coordination and control of multiple UGVs. Various methods based on artificial potential fields have been developed utilizing decoupled planning, control and prioritization in [15, 16, 17]. Roadmap theory also has tackled multi-vehicle scenarios. Optimal motion planning for multiple robots is presented, in [37], by defining a state space that simultaneously represents the configurations of all of the robots. The SBL planner [24] has been proven to be more reliable at centralized planning than decoupled planning. Experimental validation was achieved on a spot-welding station with 2 of 6 robot manipulators combining 12 to 36 degrees of freedom. Approximate cell decomposition has been used for multi-vehicle path planning; the complexity of the problem is reduced by means of a hierarchical multilevel discretization using a simple navigation function [40].

The description and use of a general framework for the study of multiple vehicle, time-coordinated path following (TC-PF) control problems is shown in [69, 70]. The framework involves generating deconflicted trajectories for the vehicles; then path following for each vehicle along its assigned path; and coordination of the relative motion of the vehicles along their paths, so as to guarantee deconfliction and meet desired temporal constraints such as equal times of arrival [70]. A survey of recent research in cooperative control of multi-vehicle systems covering the last two decades is presented in [71]. The Caltech Multi-Vehicle Wireless Testbed (MVWT) is introduced in [72]. It is a platform for testing decentralized control methodologies for multiple vehicle coordination and formation stabilization [72].

Mult-vehicle trajectory planning has many descriptive terms and categories. It has been described in the literature as centralized, decentralized, decoupled, coordinated, cooperative and/or prioritized. Centralized planning involves utilizing one state space that describes all the vehicles together as one system. Decentralized (also referred to as decoupled) planning involves path planning of each vehicle individually, thus reducing the complexity and size of the state space. Coordinated and cooperative planning are interchangeable and refer to the amount of information each vehicle knows about each other when decoupled planning is being performed. Of course, centralized planning could also be called fully cooperative, since the vehicles are in one state space and operate synergistically. Prioritization can be inserted in decoupled planning and refers to which vehicles have priority over others.

This chapter extends the implementation of the PS optimal control-based algorithm developed in this work to the autonomous trajectory planning and control of several vehicles. It uses numerous multi-vehicle scenarios, including controls that are centralized, decoupled, cooperative and prioritized. The quality of the solutions and the performance of the algorithm are studied under varying levels of information. It should be noted here that the amount of cooperation that can be achieved among vehicles depends on the amount of information that is available. For example, a vehicle cannot utilize other vehicles' course and speed data if that information is not being shared, unless the vehicle has the ability to compute course and speed based on successive environment snapshots. A vehicle algorithm may choose to use none or all the information made available. For purposes of this work, the cooperation level will be the highest obtainable for the level of information that is available. This means that the cooperation and information levels can be referred to synonymously. The communications architecture to support each level of cooperation is assumed.

B. DECOUPLED, NON-PRIORITIZED PLANNING

The decoupled approach allows each vehicle to plan its own trajectory using the information available to it, i.e., the vehicles operate independently. There is no prioritization in these scenarios; meaning that no one vehicle has priority over the others. The same three levels of information that were used in Chapter V (snapshots, prediction, *a priori*) will be used here. Also, there are three scenarios used here to study this approach: Four corners, narrow passage, and sliding door.

The algorithm developed in Chapter IV was designed for one vehicle operating in an environment of obstacles (both static and dynamic). Each vehicle's trajectory planning algorithm runs as in Figure 98 when dealing with obstacles. However, obstacles (even the dynamic ones) are not expected to change course and speed erratically. Since vehicles can alter their entire trajectories at any moment, they are treated differently. When another vehicle is encountered, it is modeled like a circular obstacle and is added to the cost and constraint equations the same way as any other obstacle is added. Following the same procedure used to develop Equation (38), the path constraint of a vehicle located at (x_v, y_v) becomes:

$$h_i(x(t), y(t)) = \ln\left(\left(\frac{x(t)-x_v}{1.0}\right)^2 + \left(\frac{y(t)-y_v}{1.0}\right)^2\right) \geq 0 \tag{64}$$

The difference is not in the problem formulation, but in the algorithm's post processing steps. When dealing with other vehicles, tight passage and narrow passage checks are not performed. Those checks were not designed to be performed on vehicles, because vehicles are continuously changing their position. A tight clearance between vehicles or between an obstacle and another vehicle is handled by modeling the vehicle properly. If a certain clearance is desired between vehicles, the radius of the circle (the *1.0* in the denominator of Equation (64)) that models the vehicle is simply increased to add the desired buffer. This is controlled by the user prior to sending a vehicle on its mission. Therefore, distinguishing between solid and grazing collisions with other vehicles is not necessary, because there is no tuning of the running cost of other vehicles. If any collision (solid or grazing) with another vehicle is detected in the danger zone, the vehicle must stop, re-position, and replan. The new flow path with the addition of collision checks with other vehicles is shown in Figure 126.

The above logic works well when using prediction or *a priori* levels of information. However, when using snapshots, a collision in the danger zone turns out to be an actual collision. For this reason, the buffer added to vehicles must be large enough to allow the vehicle to stop in time to avoid an actual collision when the algorithm detects it. For this reason, whenever the cooperation level uses snapshots, all vehicles are modeled with a buffer around them whose size is determined by the vehicle's maximum speed ($v_{\max}$), deceleration ($a_{\min}$) and time to stop (*t*):

$$buffer = \frac{1}{2} a_{\min} t^2 + v_{\max} t \tag{65}$$

This ensures no actual collision will occur when detected by the vehicle. The vehicle model used in this work requires a minimum *0.5m* radius for zero buffer. The radius is extended to *1m* when using snapshots to add enough buffer to account for its stopping distance. It should be noted here that any buffer added to the vehicles is applied in the cost function, the path constraints and collision checks when addressing vehicle-to-vehicle. Vehicle-to-obstacle equations and collision checks use the *0.5m* radius. Vehicle-to-obstacle requires no buffer since the planner controls vehicle-to-obstacle clearance through the robustness portion of the cost function.

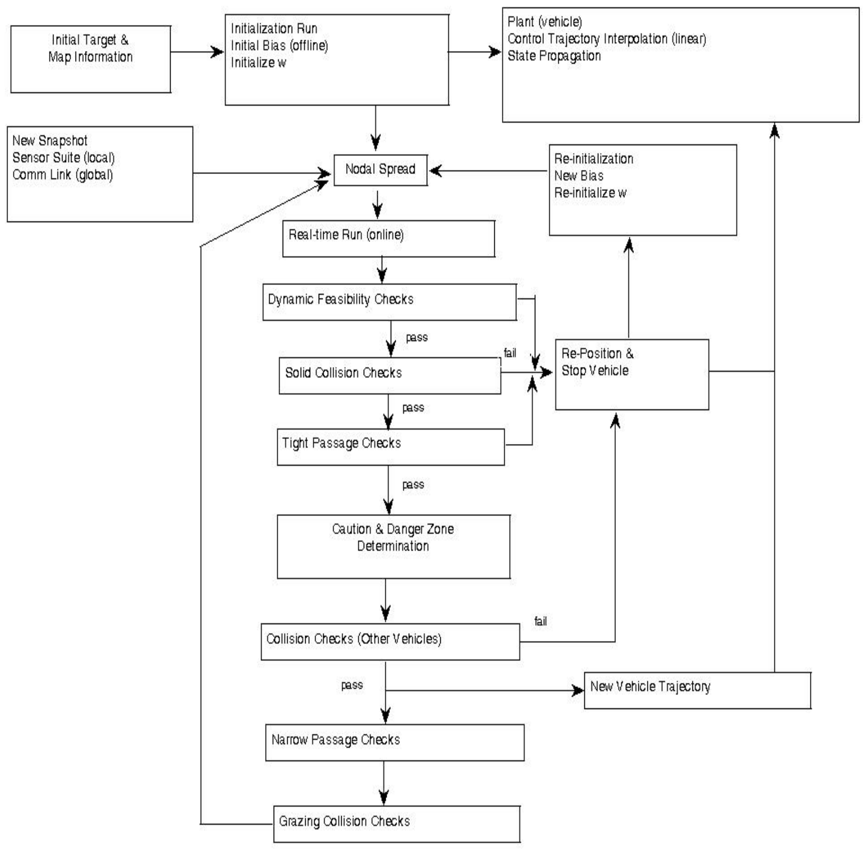


Figure 126. Multi-Vehicle Trajectory Planning Algorithm Flow Path.

1. Four Corners Scenario

The evolution of the four corners scenario using *a priori* is shown in Figure 127. Figures 128a and 128b show the final trajectories for snapshots and prediction respectively. Table 6 gives a summary of the pertinent data.

With *a priori* knowledge, the maneuver times are the fastest with only *2.2 sec* between the lowest and highest times. Maneuver times are higher and more spread out for the other cooperation levels, with prediction having the highest (furthest from optimal) times. The high maneuver times can be attributed to the fact that both snapshots and prediction needed eight replans to complete the maneuver. The average time for snapshots was faster than prediction because most of the replans were limited to vehicle 1, and were successive in nature, thus the vehicle did not waste time slowing down to a stop at each iteration since it was already stopped. The six replans of vehicle 1 can be categorized as three infeasibilities and three collision detections of its trajectory with vehicle 4. The only replan suffered by the *a priori* method involved a detected infeasibility on vehicle 2's trajectory, which happened on the very first run of the maneuver.

Average run times were all well under the *50 sec* limit set in Chapter III.D.3, however, on a few occasions, the run times exceeded *50 sec*. Although the number of runs exceeding *50 sec* is low (just over 1%), it supports the speculation made earlier that run times will always require improvement as scenarios get increasingly more complex.

In the four corners scenario, a review of the figures and data shows the performance using *a priori* was best, followed by snapshots, and then prediction. One last note regarding the four corners scenario is the fact that the vehicle start positions are displaced from the finish positions. This displacement was chosen because the trajectory planning formulation has no solution when using snapshots if a vehicle's start point is located directly on the finish point of another vehicle. Prediction and *a priori* will use the same displaced start points, keeping the scenario the same for comparison purposes.

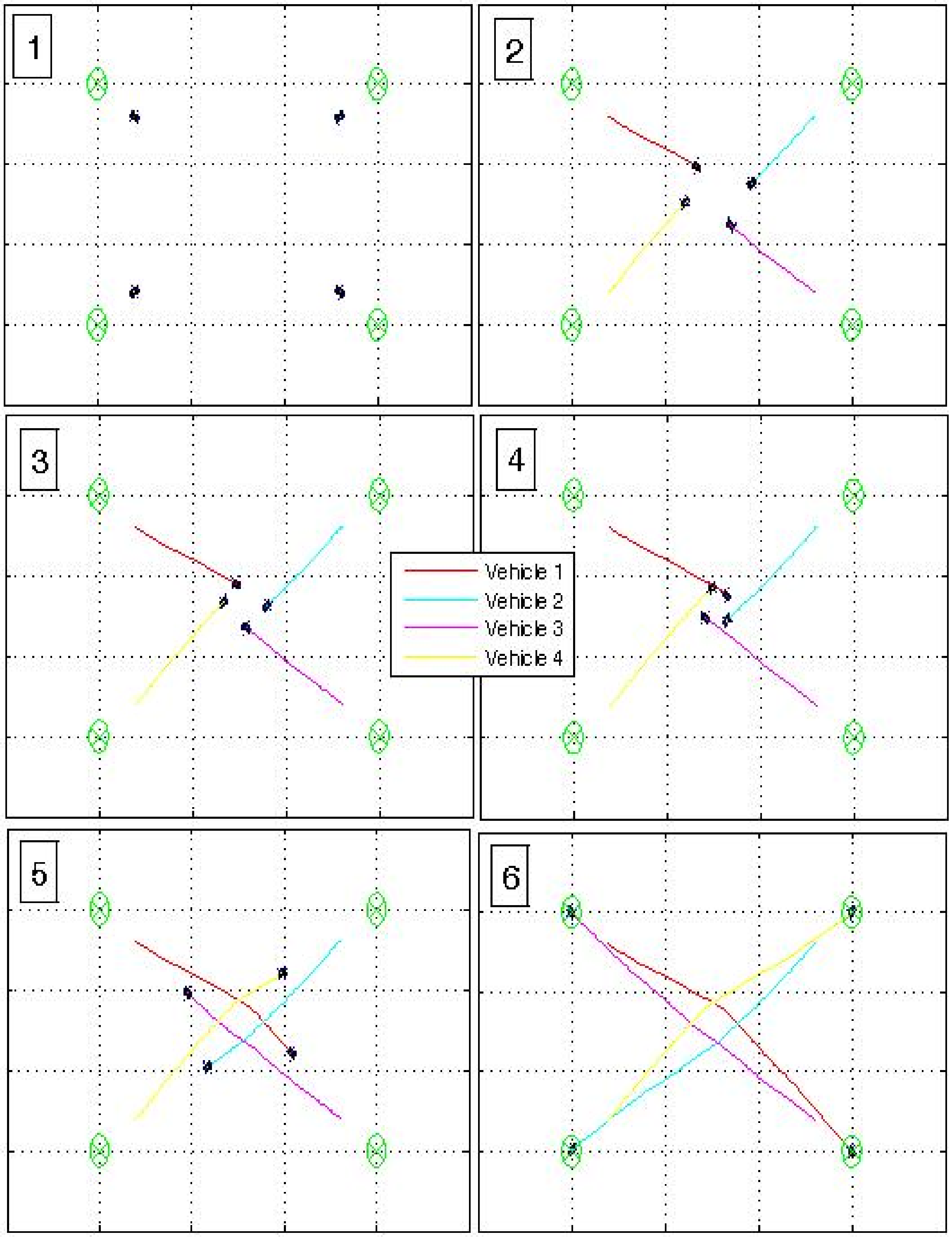


Figure 127. Evolution of the Four Corners Scenario (*a priori*).

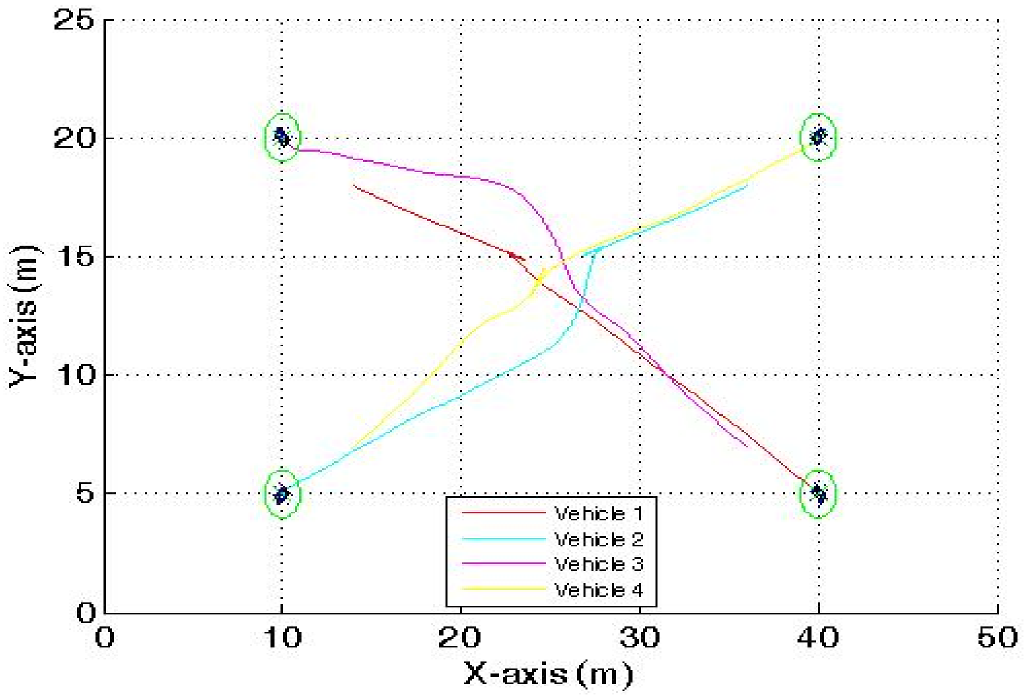


Figure 128a. Four Corners Scenario Trajectory (Snapshots).

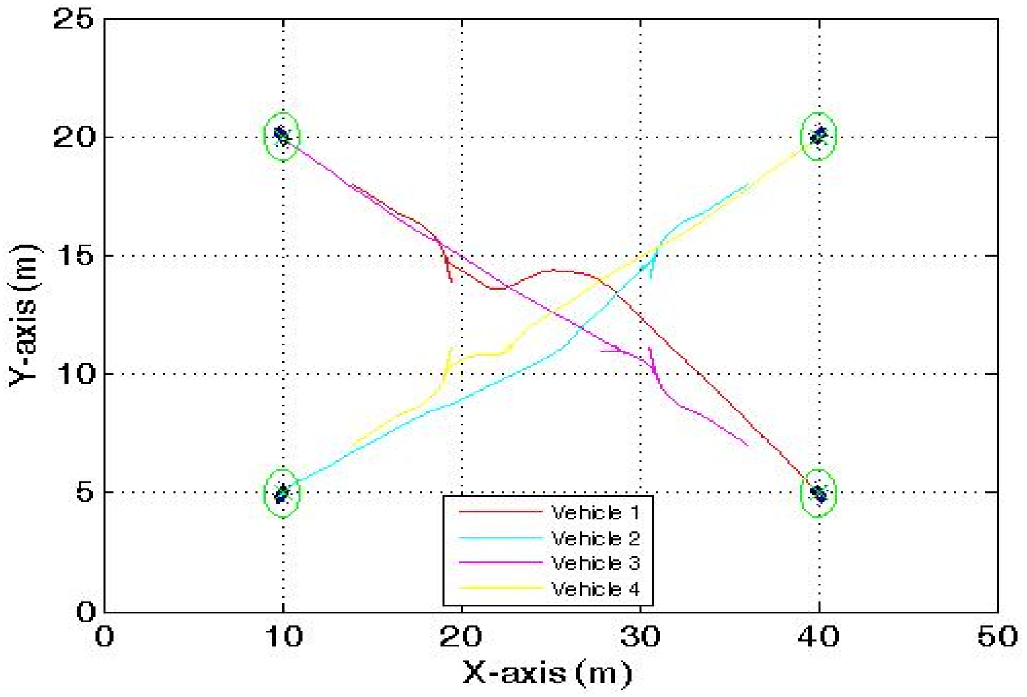


Figure 128b. Four Corners Scenario Trajectory (Prediction).

Table 6. Summary of Four Corners Scenario.

Parameter	Snapshots	Prediction	*a priori*
Maneuver Time (*s*) // Corresponding Vehicle			
Low	34.4 // 3	39.9 // 1	31.2 // 3
High	45.6 // 1	50.0 // 3	33.4 // 2
Average	39.5	45.0	31.9
Number of Re-Plans			
Vehicle 1	6	1	0
Vehicle 2	1	2	1
Vehicle 3	0	3	0
Vehicle 4	1	2	0
Average Run Time (*s*) // Runs Over *50s* (*%*)			
Vehicle 1	7.8 // 1.1	7.3 // 1.2	9.3 // 1.3
Vehicle 2	6.1 // 0.0	5.8 // 0.0	9.4 // 1.3
Vehicle 3	5.7 // 0.0	5.6 // 0.0	6.4 // 0.0
Vehicle 4	6.0 // 0.0	6.3 // 0.0	5.9 // 0.0
Min. // Max. Run Time (*s*)			
Vehicle 1	0.8 // 69.4	0.5 // 51.5	0.4 // 51.7
Vehicle 2	0.6 // 19.8	0.5 // 23.2	0.7 // 284.1
Vehicle 3	0.8 // 38.9	0.5 // 21.5	0.6 // 19.8
Vehicle 4	0.6 // 17.3	0.4 // 22.0	0.5 // 28.0

2. Narrow Passage Scenario

The evolution of the narrow passage scenario using *a priori* is shown in Figure 129. Figures 130a and 130b show the final trajectories for snapshots and prediction respectively. Table 7 gives a summary of the pertinent data.

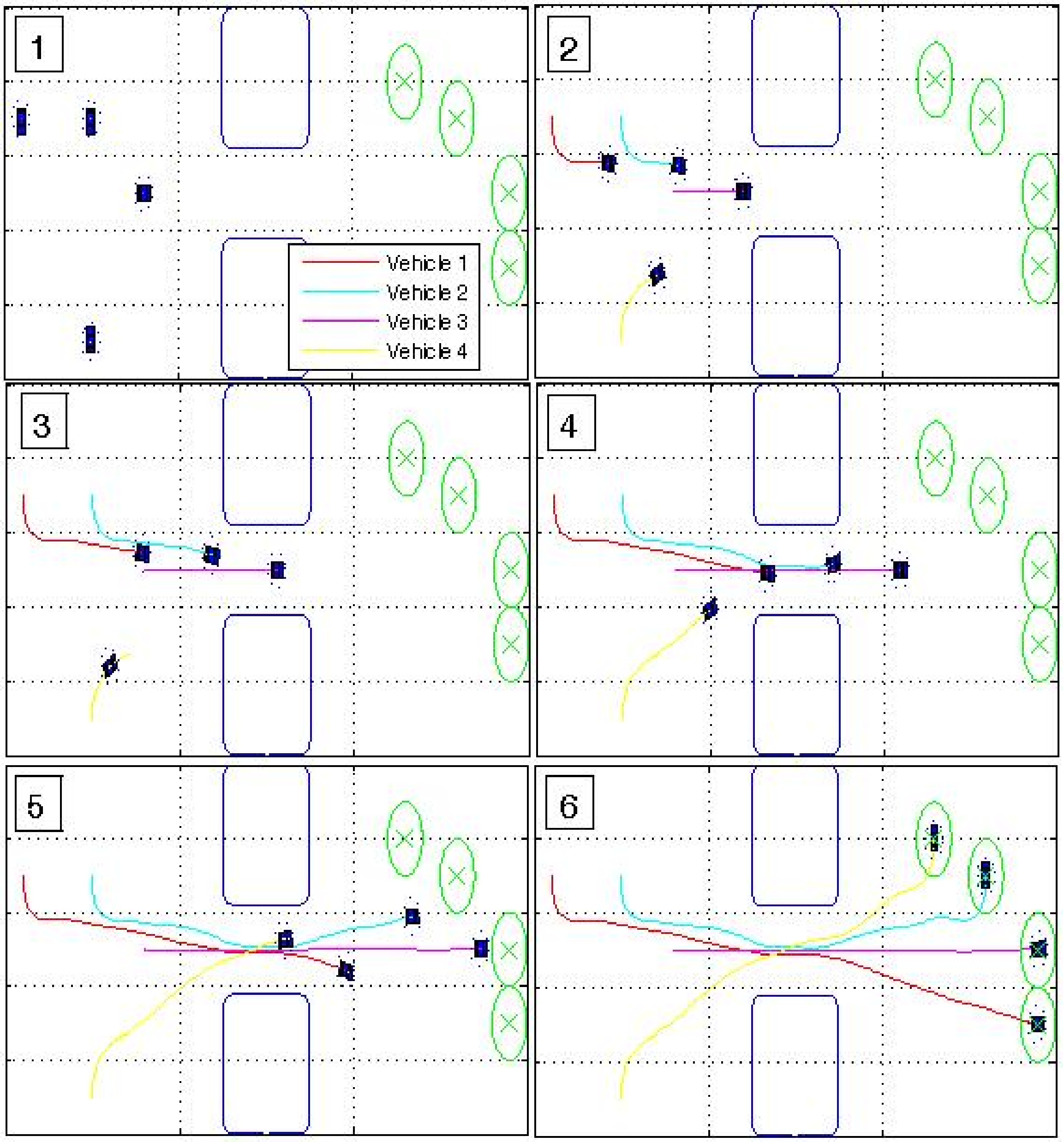


Figure 129. Evolution of the Narrow Passage Scenario (*a priori*).

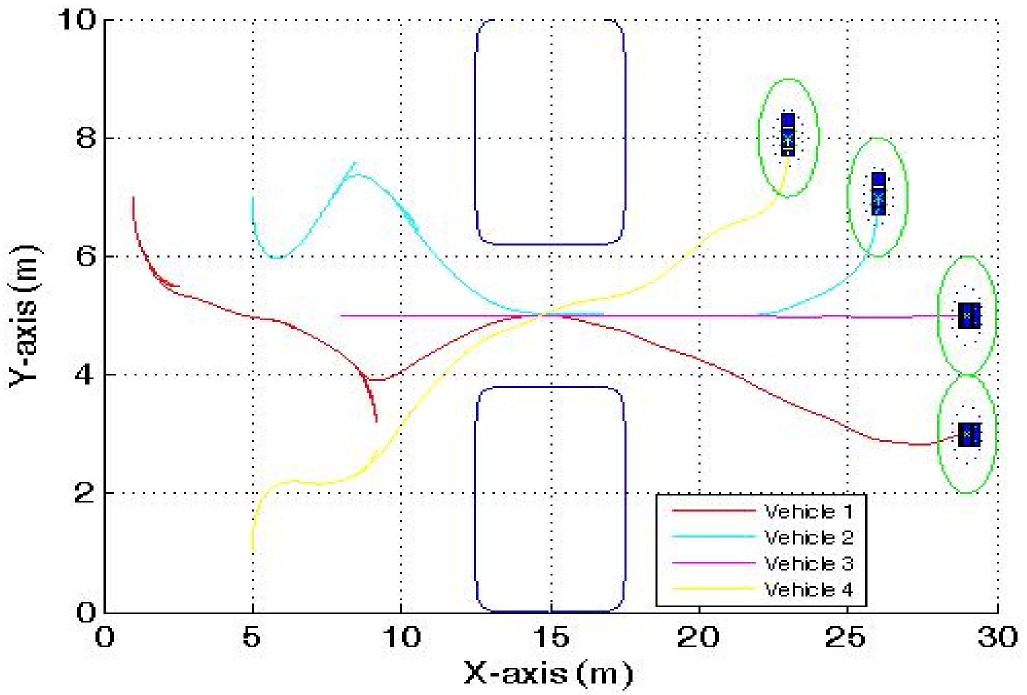


Figure 130a. Narrow Passage Scenario Trajectory (Snapshots).

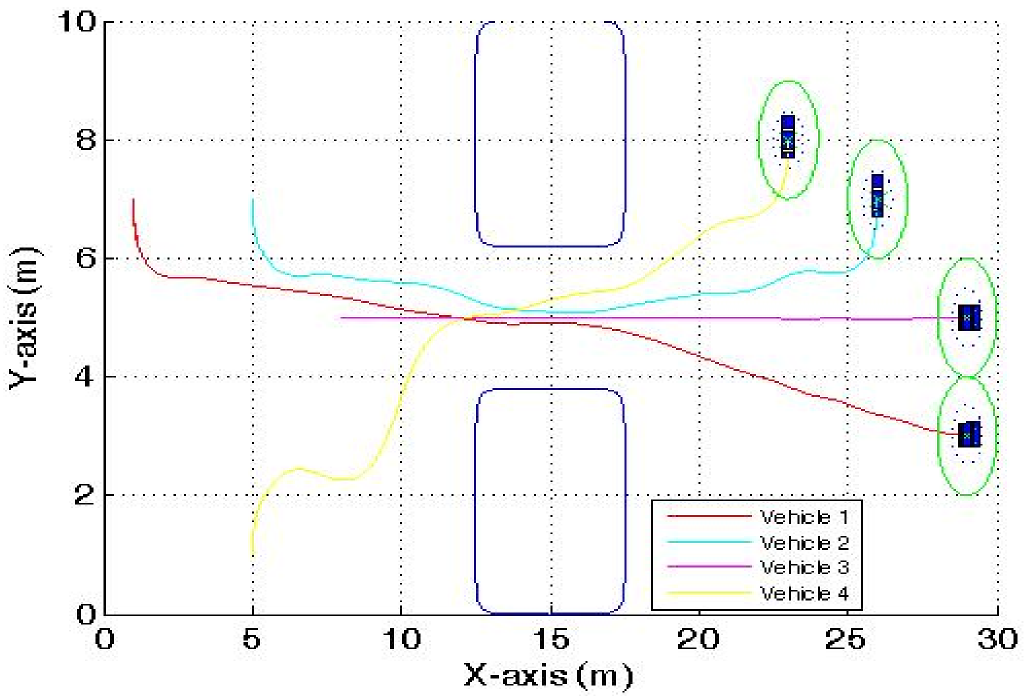


Figure 130b. Narrow Passage Scenario Trajectory (Prediction).

Table 7. Summary of Narrow Passage Scenario.

Parameter	Snapshots	Prediction	*a priori*
Maneuver Time (*s*) // Corresponding Vehicle			
Low	23.2 // 3	23.2 // 3	23.2 // 3
High	59.8 // 1	35.8 // 1	31.1 // 1
Average	39.8	26.9	27.2
Number of Re-Plans			
Vehicle 1	8	1	0
Vehicle 2	5	0	0
Vehicle 3	0	0	0
Vehicle 4	2	0	1
Average Run Time (*s*) // Runs Over *50s* (*%*)			
Vehicle 1	10.2 // 2.8	7.6 // 0.0	7.5 // 0.0
Vehicle 2	7.3 // 0.0	8.7 // 5.1	10.6 // 1.7
Vehicle 3	3.9 // 0.0	3.6 // 0.0	1.8 // 0.0
Vehicle 4	4.6 // 0.0	5.9 // 0.0	5.3 // 0.0
Min. // Max. Run Time (*s*)			
Vehicle 1	0.5 // 76.5	1.0 // 23.3	1.1 // 24.0
Vehicle 2	0.5 // 28.7	0.6 // 87.7	0.6 // 264.9
Vehicle 3	0.4 // 20.3	0.4 // 20.8	0.2 // 9.9
Vehicle 4	0.6 // 38.8	0.5 // 48.6	0.6 // 29.0

Table 7 shows that prediction and *a priori* provide similar results across the board. The biggest difference is that in prediction, vehicle 1 replans due to an infeasibility condition, whereas in *a priori* the infeasibility occurs with vehicle 4. The extra turns (using prediction) performed by vehicle 4 (see Figure 130b) at the start of the maneuver allow it to fall in behind vehicles 2 and 3 as they traverse the passage. It would have to slow down or conduct a replan if the extra turns were not used. Slowing down is contrary to the minimum time formulation, which seeks to maximize vehicle controls to complete the maneuver as fast as possible. Replanning would have worked, but no infeasibility or collision was detected to initiate a replan. In the *a priori* case, vehicle 4 does replan; having the effect of timing its passage such that it falls in behind the other vehicles. This can be seen in Figure 129, which shows that vehicle 4 has no extra turns on its way to the narrow passage. Specifically, frames 2 and 3 of Figure 129 show vehicle 4 stopping and re-positioning as it conducts its replan.

The maneuver time for vehicle 3 is the same for all three cooperation levels. This is because vehicle 3 travels to its goal, in minimum time, unimpeded by any obstacles or vehicles. Vehicle 3 will arrive at its goal in the lowest time regardless of the cooperation level. Average run times for all vehicles at each cooperation level were well under the *50 sec* limit set in Chapter III.D.3. As in the four corners case, a small percentage of run times exceeded the limit.

The snapshots case has a considerably higher average maneuver time. This is due to the fifteen replans that occur during the maneuver. Why so many replans? Since vehicle course and speeds are not used to predict their positions, they are assumed to be standing still at each environment snapshot. Any time a vehicle is in the passage, all other vehicles that have not yet traversed the passage will see it as blocked, resulting in the algorithm detecting a collision or infeasibility, which initiates a replan. The result is vehicle 3 passes through with no replans, because it is the lead vehicle and sees no blockage of the passage. Vehicle 4 replans twice as it waits for vehicle 3 to complete its transit of the passage. One of the two replans was an infeasibility, and the other was a solid collision detection of its trajectory with one of the obstacles that form the narrow passage. Vehicle 2 was the next one to pass through the passage. It experienced four infeasibilities and one collision detection with vehicle 4; all while waiting for vehicles 3 and 4 to transit the narrow passage. Finally, vehicle 1 experiences seven infeasibilities and one collision detection with vehicle 2 as it must wait for all of the other vehicles to complete their transits before it can find a valid trajectory.

In the narrow passage scenario, despite having very similar results, *a priori* is considered the better performer over prediction, because vehicle 4 follows a smoother trajectory. Snapshots comes in last here due to the long maneuver times caused by the large number of replans during the maneuver.

3. Sliding Door Scenario

The evolution of the sliding door scenario using *a priori* is shown in Figure 131. Frames 1 and 6 show the start and finish positions of the vehicles. Frame 2 was taken at the point in the maneuver when vehicle 3 has stopped to replan. This replan occurs because the algorithm detects a solid collision of vehicle 3's trajectory with obstacle 2 once the obstacle starts moving north. Frame 3 corresponds to the time when vehicle 1 stops to replan, because it detects a collision of its trajectory with vehicle 4. Frame 4 corresponds to when vehicle 2 stops to replan, because it detects a collision of its trajectory with vehicle 3. Finally, frame 5 simply shows all vehicles have clear paths to their goals with no more replans needed. Figures 132a and 132b show the final trajectories for snapshots and prediction respectively. Table 8 gives a summary of the pertinent data.

Table 8 shows again that prediction and *a priori* provide similar results across the board. The biggest difference here is that in prediction, vehicle 1 completes the maneuver with no replans, whereas in *a priori* it is vehicle 4 that conducts the maneuver with no replans. Prediction, however, suffers again from excess maneuvering to time the passage of vehicle 1, as can be seen in Figure 132b.

The snapshots case again has a considerably higher maneuver time, which can again be attributed to the high number of replans. In this case only five of the eleven replans were due to blockage of the passage. The remaining six were simply due to collision detections with other vehicles, which can be attributed to the low level of cooperation. Another observation to make from Figure 132a is the fact that the trajectory of vehicle 1 is drawn over the top of vehicle 3. This string effect is a major disadvantage that was discussed in Chapter V.B.3 and shown graphically in Figure 109.

Average run times for all vehicles at each cooperation level were again well under the *50 sec* limit set in Chapter III.D.3. In the prediction and *a priori* cases, a small percentage of run times exceeded the limit.

In the sliding door scenario, despite having very similar results, *a priori* is again considered the better performer over prediction due to the smoother trajectories. Snapshots comes in last due to the string effect on vehicle 1, and to the long maneuver times caused by the large number of replans during the maneuver.

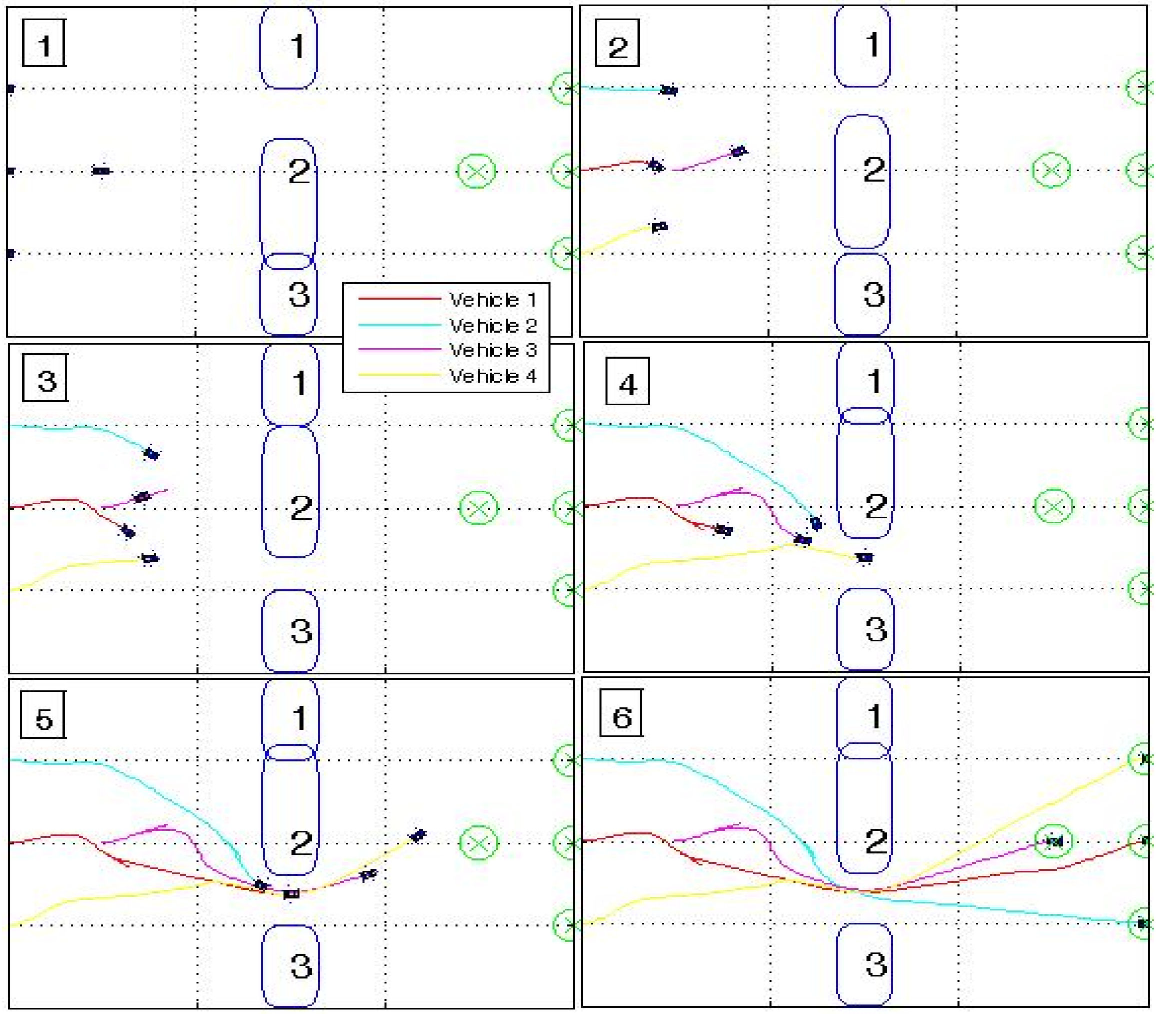


Figure 131. Evolution of the Sliding Door Scenario (*a priori*).

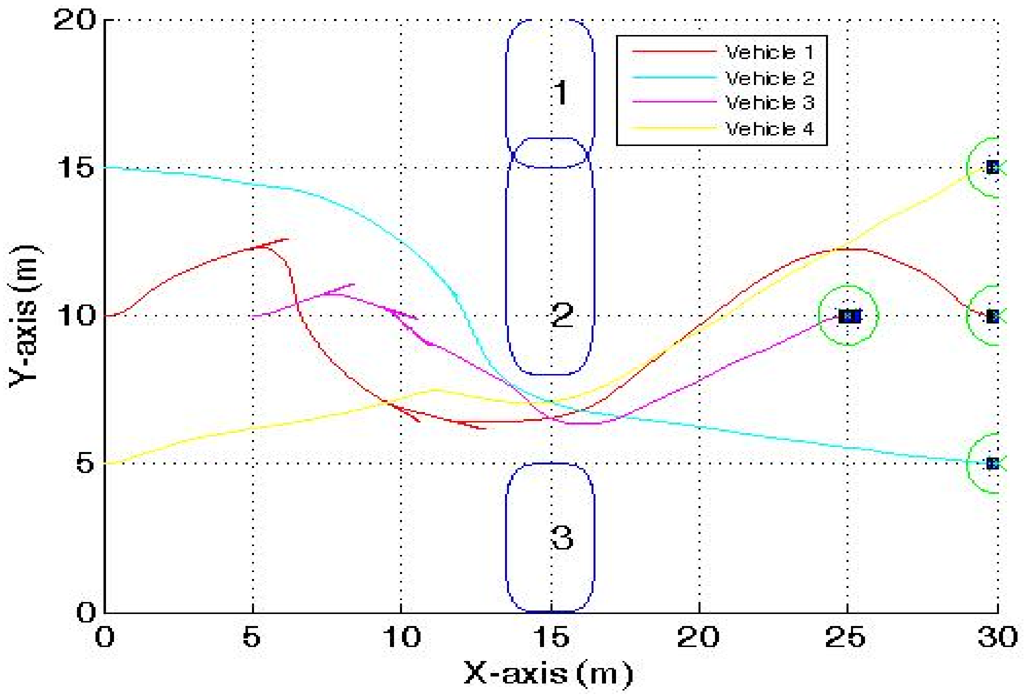


Figure 132a. Sliding Door Scenario Trajectory (Snapshots).

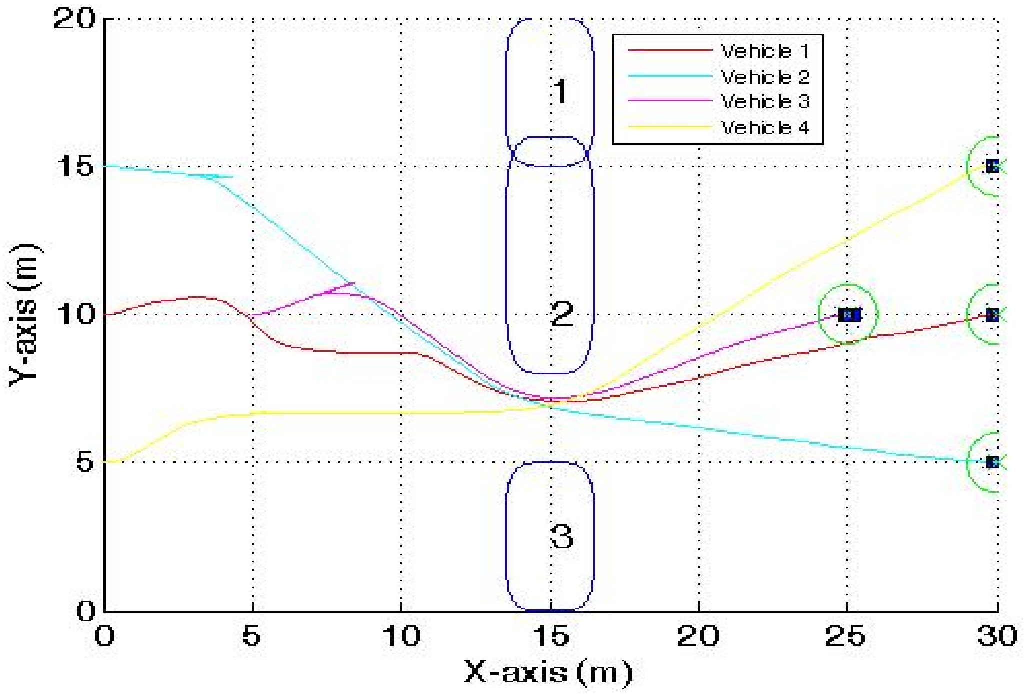


Figure 132b. Sliding Door Scenario Trajectory (Prediction).

Table 8. Summary of Sliding Door Scenario.

Parameter	Snapshots	Prediction	*a priori*
Maneuver Time (*s*) // Corresponding Vehicle			
Low	34.5 // 4	30.8 // 3	31.6 // 3
High	64.3 // 1	41.8 // 2	42.2 // 2
Average	47.8	36.9	37.7
Number of Re-Plans			
Vehicle 1	6	0	1
Vehicle 2	1	1	1
Vehicle 3	4	1	1
Vehicle 4	0	1	0
Average Run Time (*s*) // Runs Over *50s* (*%*)			
Vehicle 1	8.4 // 0.0	14.4 // 5.0	16.2 // 5.8
Vehicle 2	9.0 // 0.0	12.6 // 3.4	25.1 // 15.6
Vehicle 3	5.3 // 0.0	4.4 // 0.0	5.2 // 0.0
Vehicle 4	6.8 // 0.0	13.2 // 3.4	11.9 // 4.8
Min. // Max. Run Time (*s*)			
Vehicle 1	0.5 // 35.0	1.0 // 274.4	1.1 // 327.4
Vehicle 2	0.4 // 36.8	0.6 // 118.6	0.6 // 281.0
Vehicle 3	0.6 // 17.5	0.4 // 14.3	0.2 // 22.6
Vehicle 4	0.8 // 27.3	0.5 // 353.8	0.6 // 61.0

C. DECOUPLED, PRIORITIZED PLANNING

The same scenarios of Chapter VII.B were repeated, but in each case one of the vehicles was given priority over the others. The priority vehicle conducts its maneuver by trajectory planning with no regard to other vehicles. Its problem formulation does not consider the other vehicles in any way, which results in a solution that is completely independent of other vehicle positions or trajectories. This does not mean that the priority vehicle does not still cooperate in some small sense with the other vehicles. The algorithm developed in this work is for trajectory planning, which means that it will solve for a trajectory that gets a vehicle from start to goal while avoiding obstacles and vehicles. It does not provide for the monitoring of other vehicles for the purpose of getting out of another vehicle's way. For example, if a vehicle is stopped (either because it is conducting a replan, or it has reached its goal), it will not move to allow a priority vehicle to pass its position. For this reason, even though the priority vehicle does not consider other vehicles in its trajectory solution, it still considers other vehicles in its collision checks. If the priority vehicle senses a collision between its trajectory and another vehicle within its danger zone, the priority vehicle will react. The reaction is not to stop and replan as other vehicles would, but to simply insert the vehicle of interest into the problem formulation for the next iteration of the algorithm. This has the effect of including the vehicle of interest in the priority vehicle's next planned trajectory, so as to avoid a collision. To wit, the priority vehicle plans trajectories without regard to other vehicles, unless another vehicle poses an imminent danger, in which case the priority vehicle will conduct the appropriate maneuver to avoid collision.

1. Four Corners Scenario (Snapshots)

The evolution of the four corners scenario using snapshots, with vehicle 1 having priority, is shown in Figure 133. Table 9 gives a summary of the pertinent data, including the data obtained for the non-prioritized case (for comparison).

Vehicle 1 was prioritized because it had the highest maneuver time and the most replans. The result is vehicle 1 having no replans and the lowest maneuver time. The other reason vehicle 1 was given priority is due to the fact that it had the highest average run times and exceeded the *50 sec* limit on a few occasions. The result is vehicle 1 having the lowest average run times and no times above *50 sec*. Figure 133 shows vehicle 1 following a straight line trajectory to its goal, while all other vehicles stay out of its way.

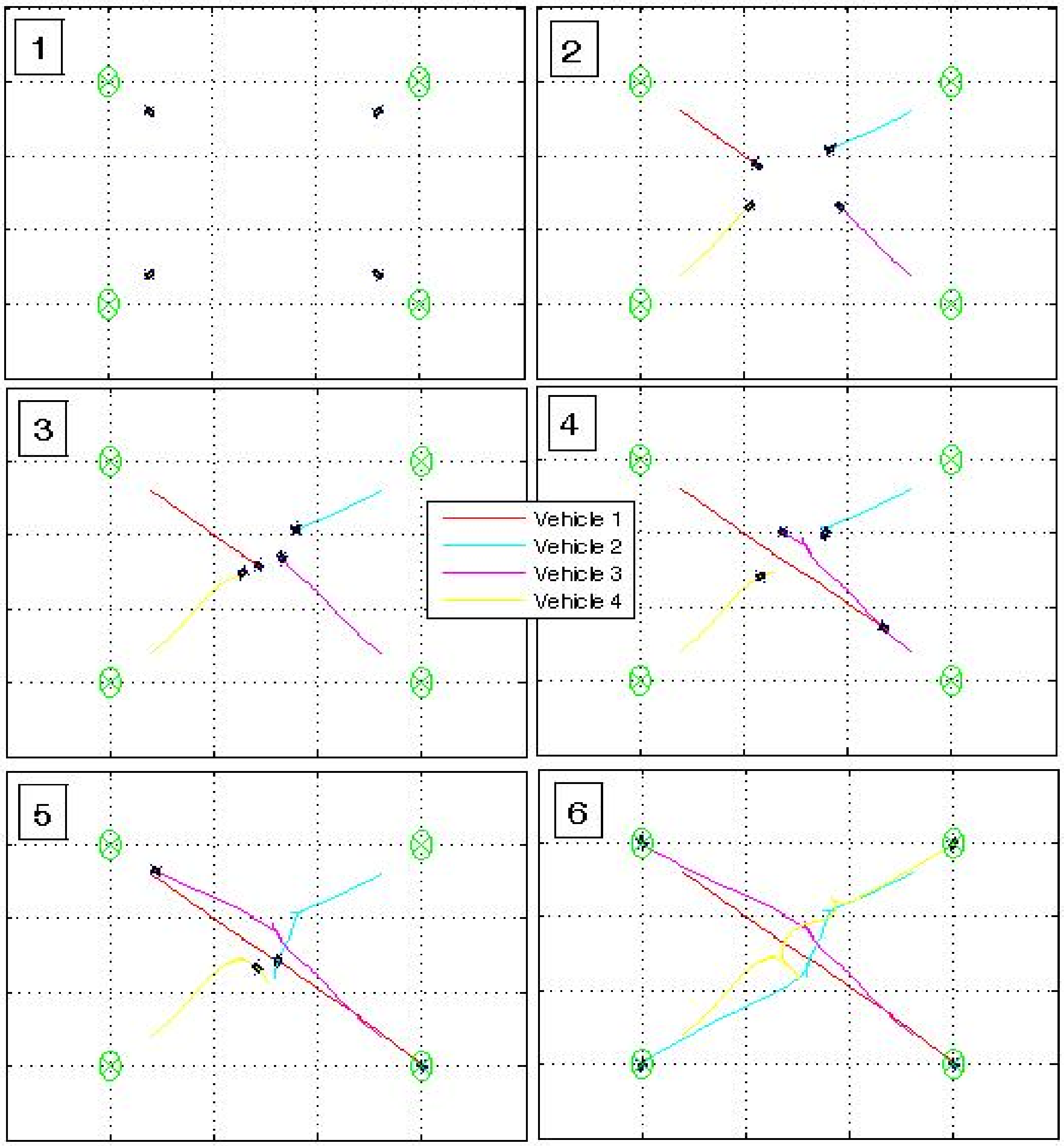


Figure 133. Evolution of Four Corners Scenario (Vehicle 1 Prioritized // Snapshots).

Table 9. Prioritized vs. Non-prioritized (Four Corners // Snapshots).

Parameter	Non-Prioritized	Vehicle 1 Prioritized
Maneuver Time (*s*) // Corresponding Vehicle		
Low	34.4 // 3	31.2 // 1
High	45.6 // 1	68.6 // 4
Average	39.5	50.8
Number of Re-Plans		
Vehicle 1	6	0
Vehicle 2	1	14
Vehicle 3	0	1
Vehicle 4	1	11
Average Run Time (*s*) // Runs Over *50s* (*%*)		
Vehicle 1	7.8 // 1.1	3.1 // 0.0
Vehicle 2	6.1 // 0.0	5.8 // 0.0
Vehicle 3	5.7 // 0.0	5.2 // 0.0
Vehicle 4	6.0 // 0.0	6.6 // 0.0
Min. // Max. Run Time (*s*)		
Vehicle 1	0.8 // 69.4	0.4 // 11.6
Vehicle 2	0.6 // 19.8	0.4 // 37.8
Vehicle 3	0.8 // 38.9	0.6 // 15.4
Vehicle 4	0.6 // 17.3	0.8 // 39.5

While prioritizing vehicle 1 improved its performance greatly, it does nothing for the performance of the other vehicles. Prioritizing one vehicle may or may not help the performance of the other vehicles. In this scenario, the performance of the other vehicles was degraded, especially for vehicles 2 and 4, which end up frozen in their tracks while trying to find trajectories around each other. Frame 5 of Figure 133 shows vehicles 2 and 4 in their frozen positions. They each continually replan, sense collisions with each other, and replan again. The two vehicles would have remained frozen this way indefinitely if not for the fact that they were given a guess to alter their decision-making process to avoid collision and allow the maneuvers to be completed. This situation of two or more vehicles being frozen in a collision detection and replan situation is another disadvantage of using snapshots for trajectory planning.

2. Four Corners Scenario (Prediction)

The evolution of the four corners scenario using prediction, with vehicle 3 having priority, is shown in Figure 134. Table 10 gives a summary of the pertinent data, including the data obtained for the non-prioritized case (for comparison).

Vehicle 3 was prioritized because it had the highest maneuver time and the most replans. The result is vehicle 3 having no replans and the lowest maneuver time. Vehicle 1's percentage of run times over *50 sec* was reduced to zero as well. Figure 134 shows vehicles 2 and 4 staying out of the way of vehicle 3, but does not show a straight trajectory for vehicle 3 to its goal. This is because vehicle 3 senses a collision within its danger zone between its trajectory and vehicle 1, while vehicle 1 is stopped and replanning. This results in vehicle 3, despite being the priority vehicle, conducting a maneuver to avoid colliding with vehicle 1.

The priority vehicle again had significantly improved performance. Despite vehicle 1 having more replans and a maneuver time that went from *39.9 sec* to *43.2 sec*, using prioritization of vehicle 3 improved the average performance of the vehicles in this scenario. Average maneuver time dropped from *45 sec* to *36.2 sec*, and the total number of replans dropped from eight to five.

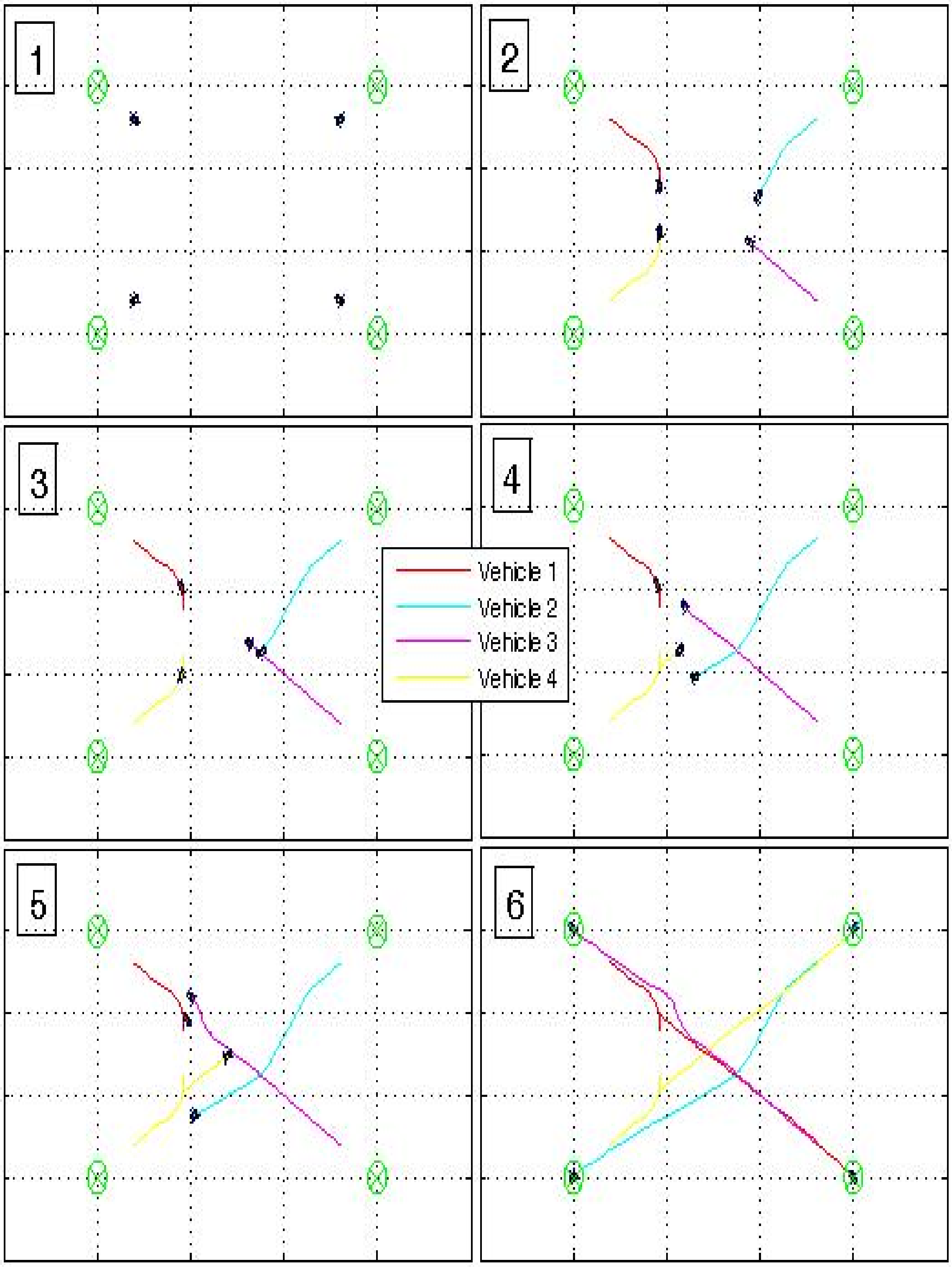


Figure 134. Evolution of Four Corners Scenario (Vehicle 3 Prioritized // Prediction).

Table 10. Prioritized vs. Non-prioritized (Four Corners // Prediction).

Parameter		Non-Prioritized	Vehicle 3 Prioritized
Maneuver Time (*s*) // Corresponding Vehicle			
	Low	39.9 // 1	31.5 // 3
	High	50.0 // 3	43.2 // 1
	Average	45.0	36.2
Number of Re-Plans	Vehicle 1	1	4
	Vehicle 2	2	0
	Vehicle 3	3	0
	Vehicle 4	2	1
Average Run Time (*s*) // Runs Over *50s* (*%*)			
	Vehicle 1	7.3 // 1.2	7.0 // 0.0
	Vehicle 2	5.8 // 0.0	6.0 // 0.0
	Vehicle 3	5.6 // 0.0	5.9 // 0.0
	Vehicle 4	6.3 // 0.0	5.5 // 0.0
Min. // Max. Run Time (*s*)			
	Vehicle 1	0.5 // 51.5	0.6 // 41.5
	Vehicle 2	0.5 // 23.2	0.5 // 42.6
	Vehicle 3	0.5 // 21.5	0.6 // 25.1
	Vehicle 4	0.4 // 22.0	0.5 // 20.5

3. Four Corners Scenario (*a priori*)

Table 11 gives a summary of the pertinent data, including the data obtained for the non-prioritized case (for comparison). Vehicle 2 was prioritized because it had the highest maneuver time and the only replan. The result is vehicle 2 having no replan and the lowest maneuver time. Its percentage of runs over *50 sec* was eliminated as well. Although priority vehicle performance improved, average vehicle performance did not, as can be seen by the higher average maneuver time.

Table 11. Prioritized vs. Non-prioritized (Four Corners // *a priori*)

Parameter		Non-Prioritized	Vehicle 2 Prioritized
Maneuver Time (*s*) // Corresponding Vehicle			
	Low	31.2 // 3	31.2 // 2
	High	33.4 // 2	38.9 // 3
	Average	31.9	33.2
Number of Re-Plans			
	Vehicle 1	0	0
	Vehicle 2	1	0
	Vehicle 3	0	1
	Vehicle 4	0	0
Average Run Time (*s*) // Runs Over *50s* (%)			
	Vehicle 1	9.3 // 1.3	7.6 // 1.3
	Vehicle 2	9.4 // 1.3	2.9 // 0.0
	Vehicle 3	6.4 // 0.0	10.9 // 1.2
	Vehicle 4	5.9 // 0.0	8.3 // 1.3
Min. // Max. Run Time (*s*):	Vehicle 1	0.4 // 51.7	0.8 // 51.0
	Vehicle 2	0.7 // 284.1	0.3 // 11.1
	Vehicle 3	0.6 // 19.8	0.5 // 66.0
	Vehicle 4	0.5 // 28.0	0.5 // 124.4

4. Narrow Passage Scenario (Snapshots)

The evolution of the narrow passage scenario using snapshots, with vehicle 1 having priority, is shown in Figure 135. Table 12 gives a summary of the pertinent data, including the data obtained for the non-prioritized case (for comparison).

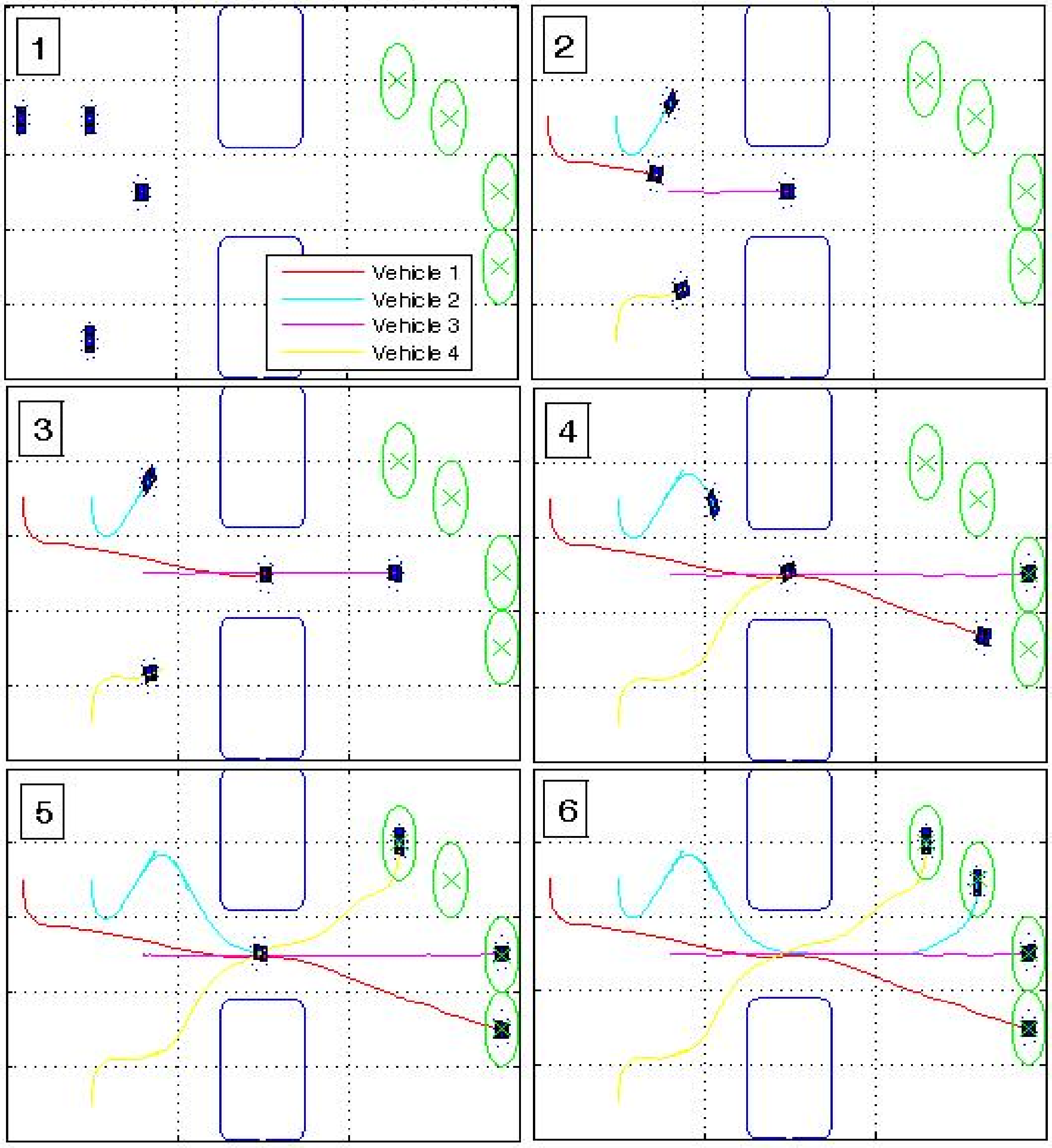


Figure 135. Evolution of Narrow Passage Scenario (Vehicle 1 Prioritized // Snapshots).

Table 12. Prioritized vs. Non-prioritized (Narrow Passage // Snapshots).

Parameter	Non-Prioritized	Vehicle 1 Prioritized
Maneuver Time (*s*) // Corresponding Vehicle		
Low	23.2 // 3	23.2 // 3
High	59.8 // 1	51.7 // 2
Average	39.8	35.9
Number of Re-Plans		
Vehicle 1	8	0
Vehicle 2	5	8
Vehicle 3	0	0
Vehicle 4	2	5
Average Run Time (*s*) // Runs Over *50s* (*%*)		
Vehicle 1	10.2 // 2.8	3.8 // 0.0
Vehicle 2	7.3 // 0.0	6.8 // 0.0
Vehicle 3	3.9 // 0.0	3.5 // 0.0
Vehicle 4	4.6 // 0.0	4.1 // 0.0
Min. // Max. Run Time (*s*)		
Vehicle 1	0.5 // 76.5	0.5 // 11.9
Vehicle 2	0.5 // 28.7	0.6 // 25.7
Vehicle 3	0.4 // 20.3	0.3 // 17.5
Vehicle 4	0.6 // 38.8	0.6 // 13.7

Vehicle 1 was prioritized because it had the highest maneuver time and the most replans. The result is vehicle 1 having no replans and the second lowest maneuver time. Run times over *50 sec* were also eliminated. Figure 135 shows vehicle 1 follows an optimal path to its goal, unimpeded by any other vehicles. It does not have to wait for vehicle 3 to clear the passage, because it follows priority vehicle rules, which allow it to plan its trajectory with no regard to other vehicles. Vehicles 2 and 4, on the other hand, undergo many replans as they can not plan valid trajectories while vehicles 1 and 3 are blocking the passage.

The overall performance of the vehicles in this scenario was improved by prioritizing vehicle 1. The average maneuver time dropped, and there were fewer replans. Average run times were lower as well.

5. Narrow Passage Scenario (Prediction)

Table 13 gives a summary of the pertinent data, including the data obtained for the non-prioritized case (for comparison). Vehicle 1 was prioritized because it had the highest maneuver time and the only replan. The result is vehicle 1 completes the maneuver in minimum time with no replan. By the numbers (Table 13), average performance of the other vehicles did not change significantly; however, the trajectories were smoother. The resulting trajectories were nearly identical to the non-prioritized (*a priori*) case in Figure 129. The extra vehicle turns shown in Figure 130b were eliminated. Comparing the data from Table 7 (non-prioritized, *a priori*) to the data in Table 13 (vehicle 1 prioritized, prediction) shows the results of the two scenarios to be very similar. It is determined here that prioritizing vehicle 1 improved the overall performance of the vehicles due to the smoother trajectories.

Table 13. Prioritized vs. Non-prioritized (Narrow Passage // Prediction).

Parameter		Non-Prioritized	Vehicle 1 Prioritized
Maneuver Time (*s*) // Corresponding Vehicle			
	Low	23.2 // 3	23.2 // 3
	High	35.8 // 1	31.0 // 1
	Average	26.9	27.1
Number of Re-Plans	Vehicle 1	1	0
	Vehicle 2	0	0
	Vehicle 3	0	0
	Vehicle 4	0	1
Average Run Time (*s*) // Runs Over *50s* (*%*)			
	Vehicle 1	7.6 // 0.0	4.0 // 0.0
	Vehicle 2	8.7 // 5.1	4.7 // 0.0
	Vehicle 3	3.6 // 0.0	4.2 // 0.0
	Vehicle 4	5.9 // 0.0	5.2 // 1.7
Min. // Max. Run Time (*s*)	Vehicle 1	1.0 // 23.3	0.6 // 11.8
	Vehicle 2	0.6 // 87.7	0.6 // 18.7
	Vehicle 3	0.4 // 20.8	0.4 // 20.5
	Vehicle 4	0.5 // 48.6	0.6 // 59.0

6. Narrow Passage Scenario (*a priori*)

The evolution of the narrow passage scenario using *a priori*, with vehicle 4 having priority, is shown in Figure 136. Table 14 gives a summary of the pertinent data, including the data obtained for the non-prioritized case (for comparison).

Vehicle 4 was prioritized because it had one replan and was the last vehicle to pass through the passage in the non-prioritized case (Figure 129). The effect is that vehicle 4 becomes the second vehicle to traverse the passage, and the replan shifts to vehicle 2, which becomes the last vehicle in the line. The trajectory of vehicle 4 was improved at the expense of vehicle 2, but the average performance across the board did not change significantly. The *a priori* case already gave the optimal solution in the non-prioritized case. Prioritizing simply changed the order that the vehicles travelled through the passage.

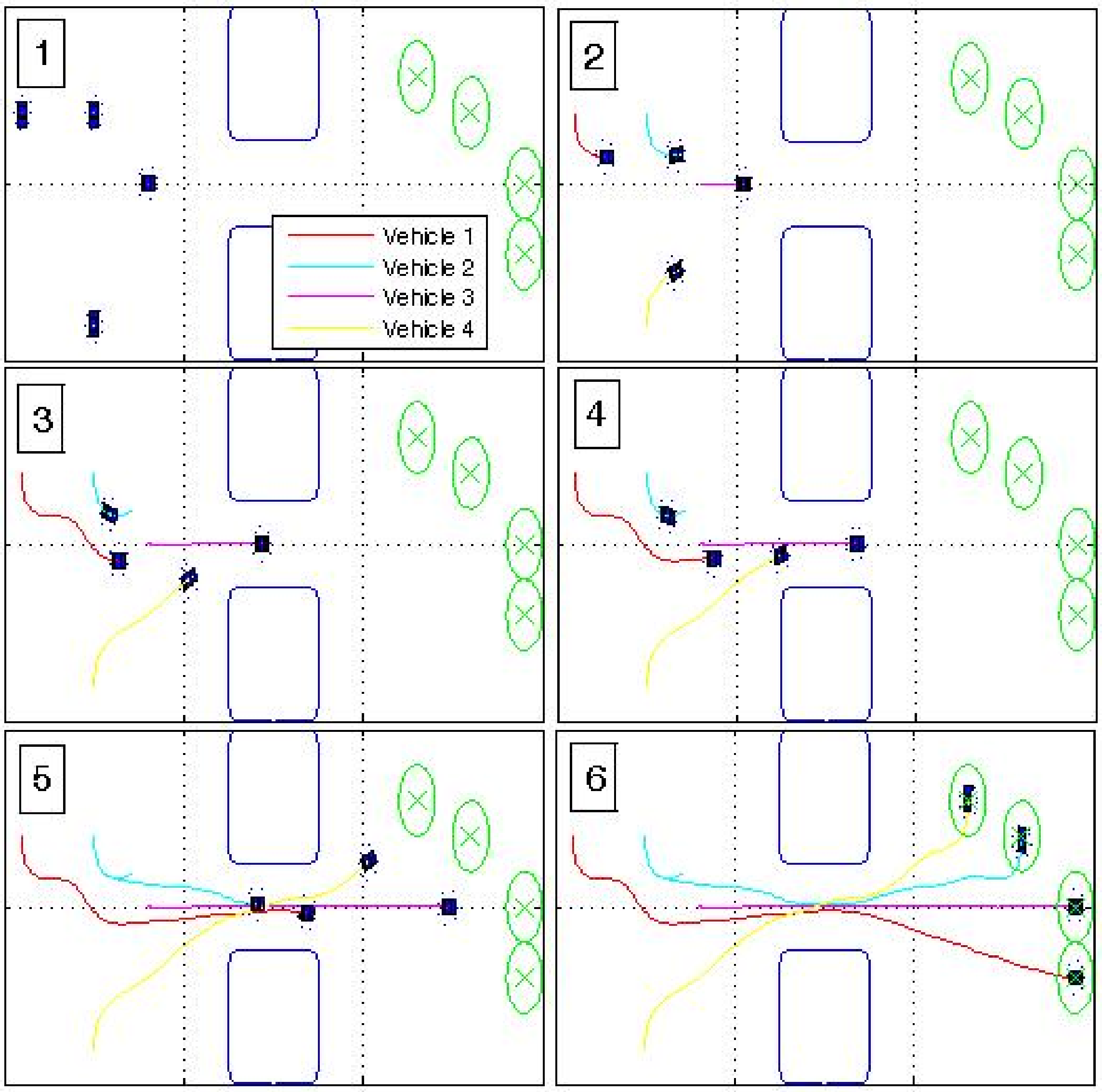


Figure 136. Evolution of the Narrow Passage Scenario (Vehicle 4 Prioritized // *a priori*).

Table 14. Prioritized vs. Non-prioritized (Narrow Passage // *a priori*).

Parameter		Non-Prioritized	Vehicle 4 Prioritized
Maneuver Time (*s*) // Corr Veh	Low	23.2 // 3	22.5 // 4
	High	31.1 // 1	31.9 // 2
	Average	27.2	27.2
Number of Re-Plans	Vehicle 1	0	0
	Vehicle 2	0	1
	Vehicle 3	0	0
	Vehicle 4	1	0
Average Run Time (*s*) // Runs Over *50s* (*%*)			
	Vehicle 1	7.5 // 0.0	7.8 // 1.3
	Vehicle 2	10.6 // 1.7	13.7 // 10.9
	Vehicle 3	1.8 // 0.0	3.5 // 0.0
	Vehicle 4	5.3 // 0.0	1.7 // 0.0
Min. // Max. Run Time (*s*)	Vehicle 1	1.1 // 24.0	0.6 // 86.3
	Vehicle 2	0.6 // 264.9	0.7 // 218.6
	Vehicle 3	0.2 // 9.9	0.3 // 21.7
	Vehicle 4	0.6 // 29.0	0.3 // 4.6

7. Sliding Door Scenario (Snapshots)

The evolution of the sliding door scenario using snapshots, with vehicle 1 prioritized, is shown in Figure 137. Table 15 gives a summary of the pertinent data, including the data obtained for the non-prioritized case (for comparison).

Vehicle 1 was prioritized because it had the highest maneuver time and the most replans. The result was vehicle 1 having the lowest maneuver time and zero replans. The effect of prioritization on the system as a whole was degraded in that the average maneuver time went up and the total number of replans was greater. Even though the average run times for three of the vehicles went down, vehicle 4 suffered an increase in run times and had a small percentage exceed *50 sec*.

Frame 2 of Figure 137 shows the priority vehicle conducting a maneuver (by design) to avoid colliding with vehicle 3, which is in the middle of replanning. In frame 3, vehicle 4 is giving way to the priority vehicle. In frame 4, vehicles 2 and 3 have to stop and replan due to vehicle 4 blocking the passage (a negative aspect of snapshots). Frames 5 and 6 show the wide turn made by vehicle 4, which again demonstrates the string effect, which was discussed in Chapter V.B.3. Essentially, vehicle 4 is drawn off its optimal trajectory by vehicle 1.

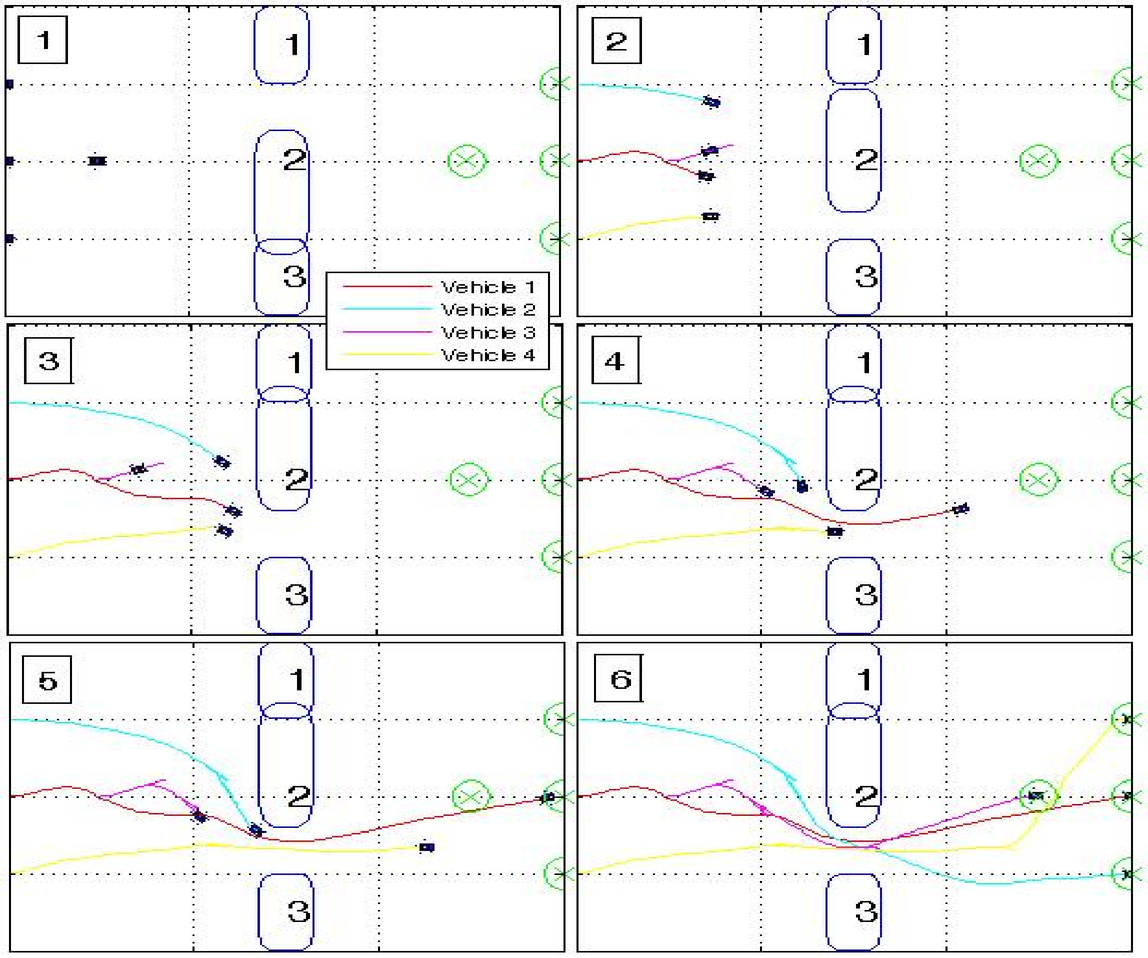


Figure 137. Evolution of the Sliding Door Scenario (Vehicle 1 Prioritized // Snapshots).

Table 15. Prioritized vs. Non-prioritized (Sliding Door // Snapshots).

Parameter		Non-Prioritized	Vehicle 1 Prioritized
Maneuver Time (*s*) // Corresponding Vehicle			
	Low	34.5 // 4	33.2 // 1
	High	64.3 // 1	60.0 // 3
	Average	47.8	48.3
Number of Re-Plans	Vehicle 1	6	0
	Vehicle 2	1	2
	Vehicle 3	4	10
	Vehicle 4	0	2
Average Run Time (*s*) // Runs Over *50s* (*%*)			
	Vehicle 1	8.4 // 0.0	7.5 // 0.0
	Vehicle 2	9.0 // 0.0	8.1 // 0.0
	Vehicle 3	5.3 // 0.0	5.2 // 0.0
	Vehicle 4	6.8 // 0.0	9.6 // 1.0
Min. // Max. Run Time (*s*)			
	Vehicle 1	0.5 // 35.0	0.6 // 23.4
	Vehicle 2	0.4 // 36.8	0.9 // 33.2
	Vehicle 3	0.6 // 17.5	0.9 // 26.4
	Vehicle 4	0.8 // 27.3	0.6 // 159.5

8. Sliding Door Scenario (Prediction)

Table 16 gives a summary of the pertinent data, including the data obtained for the non-prioritized case (for comparison). Vehicle 2 was prioritized because it had the highest maneuver time and one replan. The result was vehicle 2 having a lower maneuver time (*34.8 sec*) and no replan. The effect of prioritization improved the average maneuver time of the vehicles, and reduced the number of replans to just one. To wit, in this scenario, using prioritization improved the overall performance of the system of vehicles in addition to improving the performance of the priority vehicle.

Table 16. Prioritized vs. Non-prioritized (Sliding Door // Prediction).

Parameter		Non-Prioritized	Vehicle 2 Prioritized
Maneuver Time (*s*) // Corresponding Vehicle			
	Low	30.8 // 3	31.5 // 3
	High	41.8 // 2	38.9 // 1
	Average	36.9	35.5
Number of Re-Plans	Vehicle 1	0	0
	Vehicle 2	1	0
	Vehicle 3	1	1
	Vehicle 4	1	0
Average Run Time (*s*) // Runs Over *50s* (*%*)			
	Vehicle 1	14.4 // 5.0	16.1 // 6.4
	Vehicle 2	12.6 // 3.4	3.9 // 0.0
	Vehicle 3	4.4 // 0.0	13.5 // 1.6
	Vehicle 4	13.2 // 3.4	10.9 // 2.4
Min. // Max. Run Time (*s*)	Vehicle 1	1.0 // 274.4	0.7 // 143.0
	Vehicle 2	0.6 // 118.6	0.5 // 10.9
	Vehicle 3	0.4 // 14.3	0.8 // 524.2
	Vehicle 4	0.5 // 353.8	0.9 // 68.7

9. Sliding Door Scenario (*a priori*)

Table 17 gives a summary of the pertinent data, including the data obtained for the non-prioritized case (for comparison). Vehicle 2 was prioritized because it had the highest maneuver time and one replan. The result was vehicle 2 having a lower maneuver time (*34.8 sec*) and no replan. Note that the maneuver time for vehicle 2 in this scenario is identical to that of the prediction case, because in that case it was also vehicle 2 that was prioritized. This identical maneuver time indicates vehicle 2 uses the time optimal trajectory. In this scenario, using prioritization improved the overall performance of the system of vehicles, as indicated by the improved average maneuver time.

D. CENTRALIZED PLANNING

Centralized planning involves utilizing one state space that describes all the vehicles together as one system. It is fully cooperative, since the vehicles operate synergistically. Prioritizing a vehicle is not possible without decoupling it from the other vehicles. Collision checks between vehicles are not performed, because they are not solving separate trajectories; rather, one trajectory is determined which includes all the vehicles together as a single system. The discrete nature of the solutions may cause vehicles to graze each other if they come too close to one another. The user can prevent this (and even add more distance between vehicles as a safety measure) by simply adding a buffer around each vehicle (see Chapter VII.B). Since there are no collision checks between vehicles, the algorithm flow path follows the single vehicle logic of Figure 98 (vice the multi-vehicle flow path of Figure 126).

Table 17. Prioritized vs. Non-prioritized (Sliding Door // *a priori*).

Parameter		Non-Prioritized	Vehicle 2 Prioritized
Maneuver Time (*s*) // Corresponding Vehicle			
	Low	31.6 // 3	33.6 // 3
	High	42.2 // 2	42.5 // 4
	Average	37.7	37.0
Number of Re-Plans	Vehicle 1	1	0
	Vehicle 2	1	0
	Vehicle 3	1	2
	Vehicle 4	0	1
Average Run Time (*s*) // Runs Over *50s* (*%*)			
	Vehicle 1	16.2 // 5.8	15.9 // 6.7
	Vehicle 2	25.1 // 15.6	4.0 // 0.0
	Vehicle 3	5.2 // 0.0	7.2 // 1.6
	Vehicle 4	11.9 // 4.8	14.3 // 5.4
Min. // Max. Run Time (*s*)			
	Vehicle 1	1.1 // 327.4	0.7 // 122.5
	Vehicle 2	0.6 // 281.0	0.4 // 10.8
	Vehicle 3	0.2 // 22.6	0.5 // 87.2
	Vehicle 4	0.6 // 61.0	0.6 // 110.5

1. Preliminary Observations

Initial simulations for the narrow passage and sliding door scenarios are shown in Figures 138a and 138b, respectively. The extra turns taken by the vehicles are due to the fact that in treating the four vehicles as one system with one trajectory, there is only one time vector. One time vector has the effect of forcing all the vehicles to complete their maneuvers at the same time. Minimum time is, therefore, limited to the worst case vehicle. In the narrow passage case (Figure 138a), vehicle 1 is the limiting vehicle, and all other vehicles must loiter about until vehicle 1 reaches its goal in order for all vehicles to finish simultaneously. Table 18 gives a summary of the data for all three scenarios. Since all the vehicles finish at the same time, all the maneuver times are equal. The majority of the run times exceed the *50 sec* limit. The high run times can be attributed to two factors. First, since there is only one time vector, the minimum required number of nodes is limited to the vehicle with the longest trajectory. This means that for any vehicle requiring fewer nodes to solve its trajectory, it will be using an excess number of nodes, which unnecessarily increases the run time. The second reason for the increased run times is the higher dimensionality of the problem. When conducting decoupled trajectory planning, the state space is only five variables. But when conducting centralized planning, the state space increases to twenty variables (five for each vehicle).

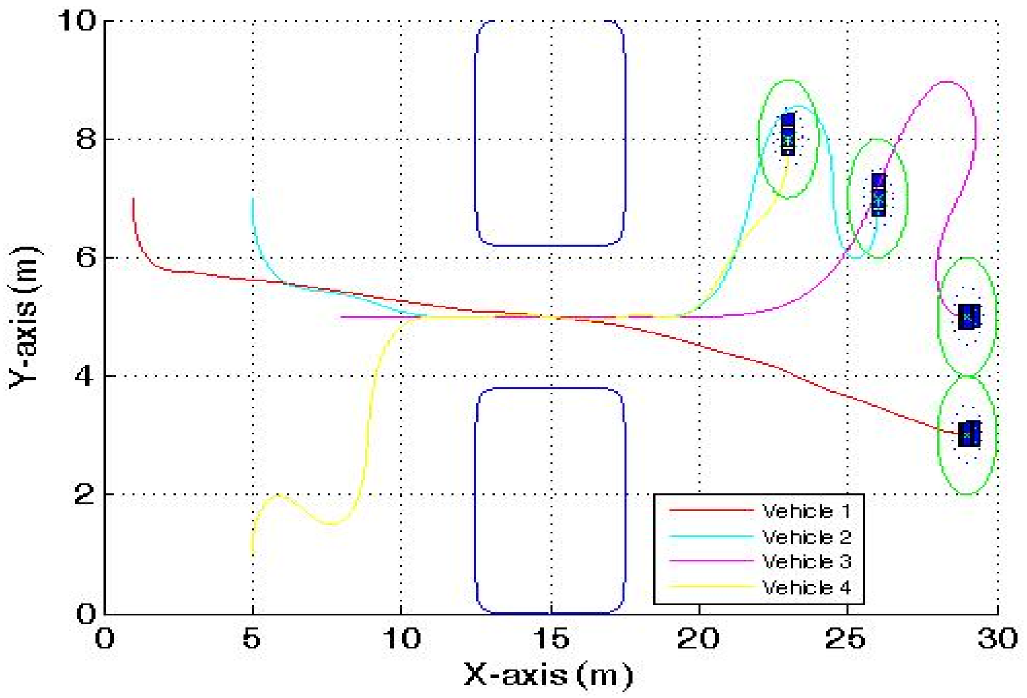


Figure 138a. Initial Simulation of Narrow Passage Scenario (Centralized Planning).

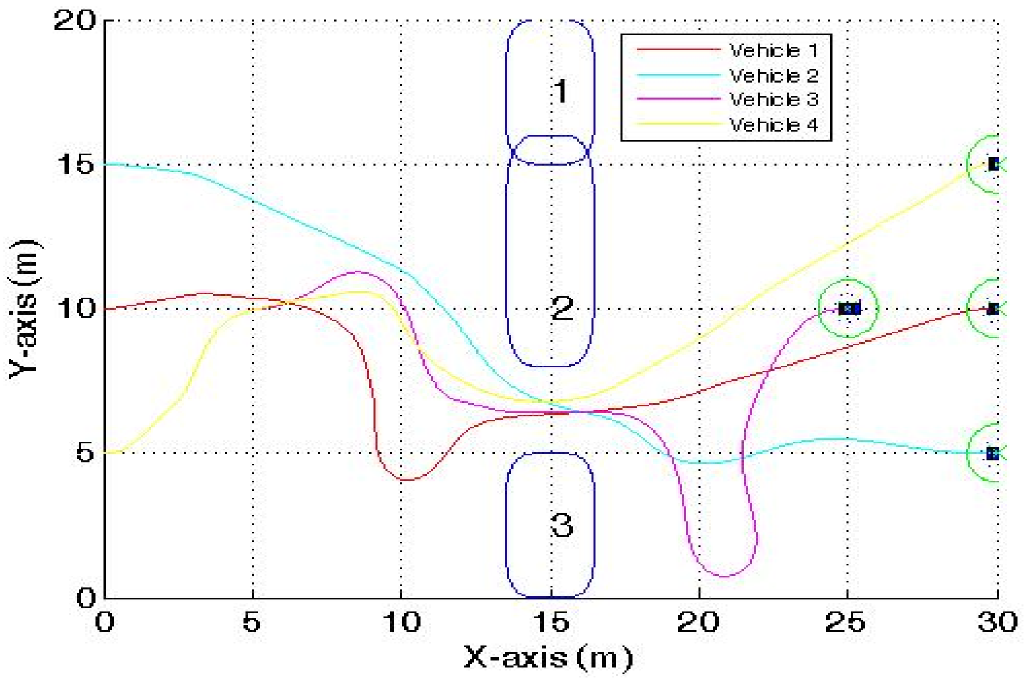


Figure 138b. Initial Simulation of Sliding Door Scenario (Centralized Planning).

Table 18. Summary of Initial Simulations using Centralized Planning.

Parameter		Four Corners	Narrow Pass	Sliding Door
Maneuver Time (*s*) // Corr Vehicle				
	Low	32.0 // all	31.2 // all	39.6 // all
	High	32.0 // all	31.2 // all	39.6 // all
	Average	32.0	31.2	39.6
Number of Re-Plans	Vehicle 1	0	0	0
	Vehicle 2	0	0	0
	Vehicle 3	0	0	0
	Vehicle 4	0	0	0
Ave Run Time (*s*) // Runs Over *50s* (*%*)		57.1 // 39.5	113.6 // 74.3	137.0 // 61.3
Min. // Max. Run Time (*s*)		3.1 // 290.7	3.7 // 802.0	0.8 // 2280.0

2. Improved Formulation

The desire here is to enable each vehicle to reach its goal in minimum time, but still use one time vector. Since minimum distance can be synonymous with minimum time, the improved trajectories can be achieved with one simple change to the problem formulation. Specifically, this change is made to the original cost function, which is repeated here as Equation (66):

$$J[\underline{x}(\cdot),\underline{u}(\cdot),t_f]=t_f+\int_0^{t_f} r(t)dt \tag{66}$$

A distance-to-goal cost is added to Equation (66). This distance-to-goal cost essentially adds to the overall cost of the trajectory when the vehicle is not yet at its goal position. The goal cost is simply a function of the vehicle's distance from its desired goal. Using the same argument for scaling and balancing that was made when developing the path following cost of Chapter VI.B, the natural log is used to properly balance the goal cost with the minimum time and robustness costs. The result is Equation (67) representing the goal cost.

$$g(x(t),y(t))=\ln\left(\left(\frac{x(t)-x_f}{0.1}\right)^2+\left(\frac{y(t)-y_f}{0.1}\right)^2\right) \tag{67}$$

The new cost equation adds the goal cost to the original minimum time/maximum robustness formulation, resulting in Equation (68) as the new formulation for the overall cost:

$$J[\underline{x}(\cdot),\underline{u}(\cdot),t_f]=t_f+\int_0^{t_f}\left(r(t)+g(t)\right)dt \tag{68}$$

3. Final Observations

The new cost formulation was used to perform the four corners, narrow passage and sliding door simulations. The evolution of each of the scenarios is shown in Figures 139, 140 and 141. Vehicle trajectories no longer show the vehicles loitering in order to finish at the same time. Each vehicle completes its maneuver in minimum time, as desired.

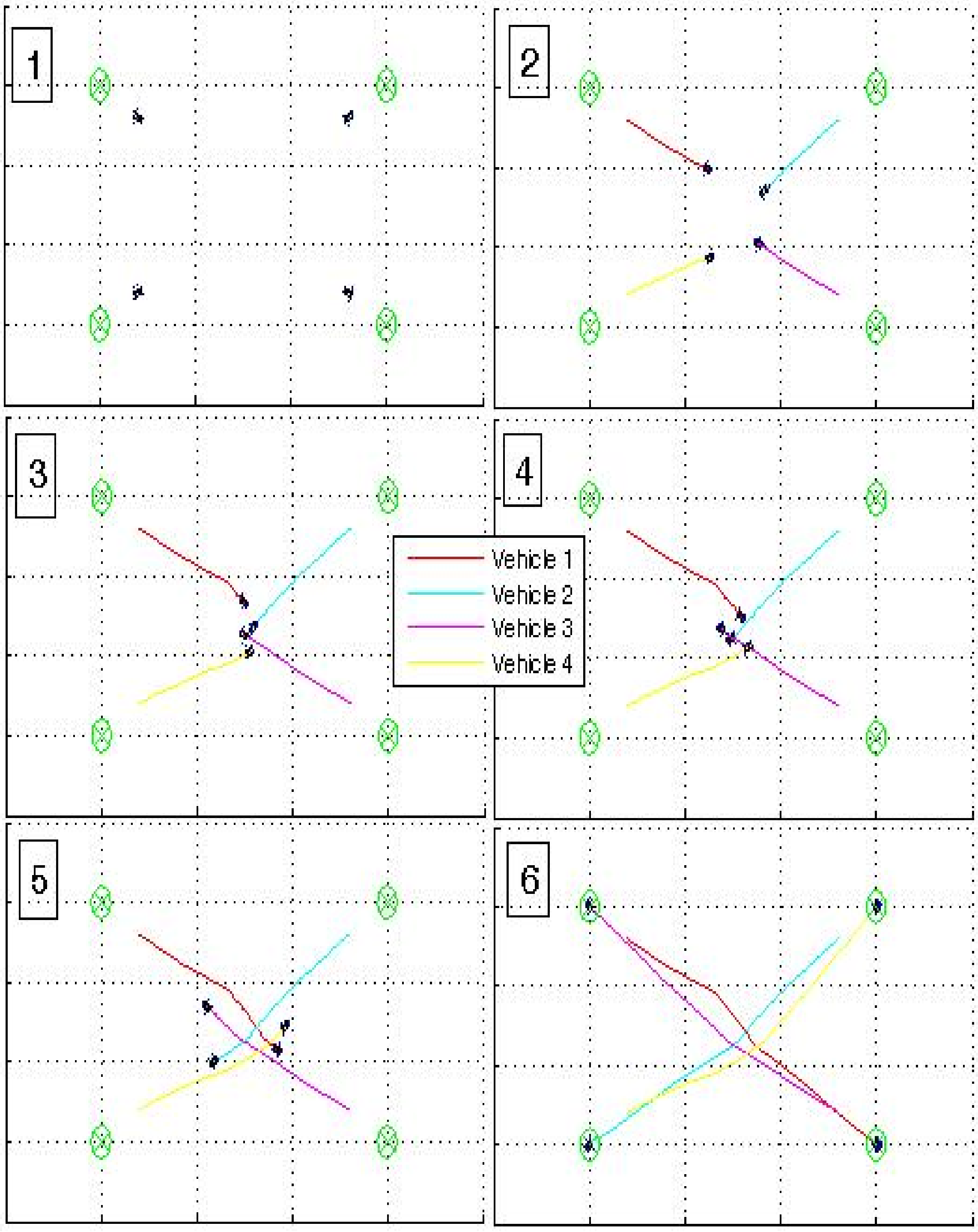


Figure 139. Evolution of the Four Corners Scenario (Centralized Planning).

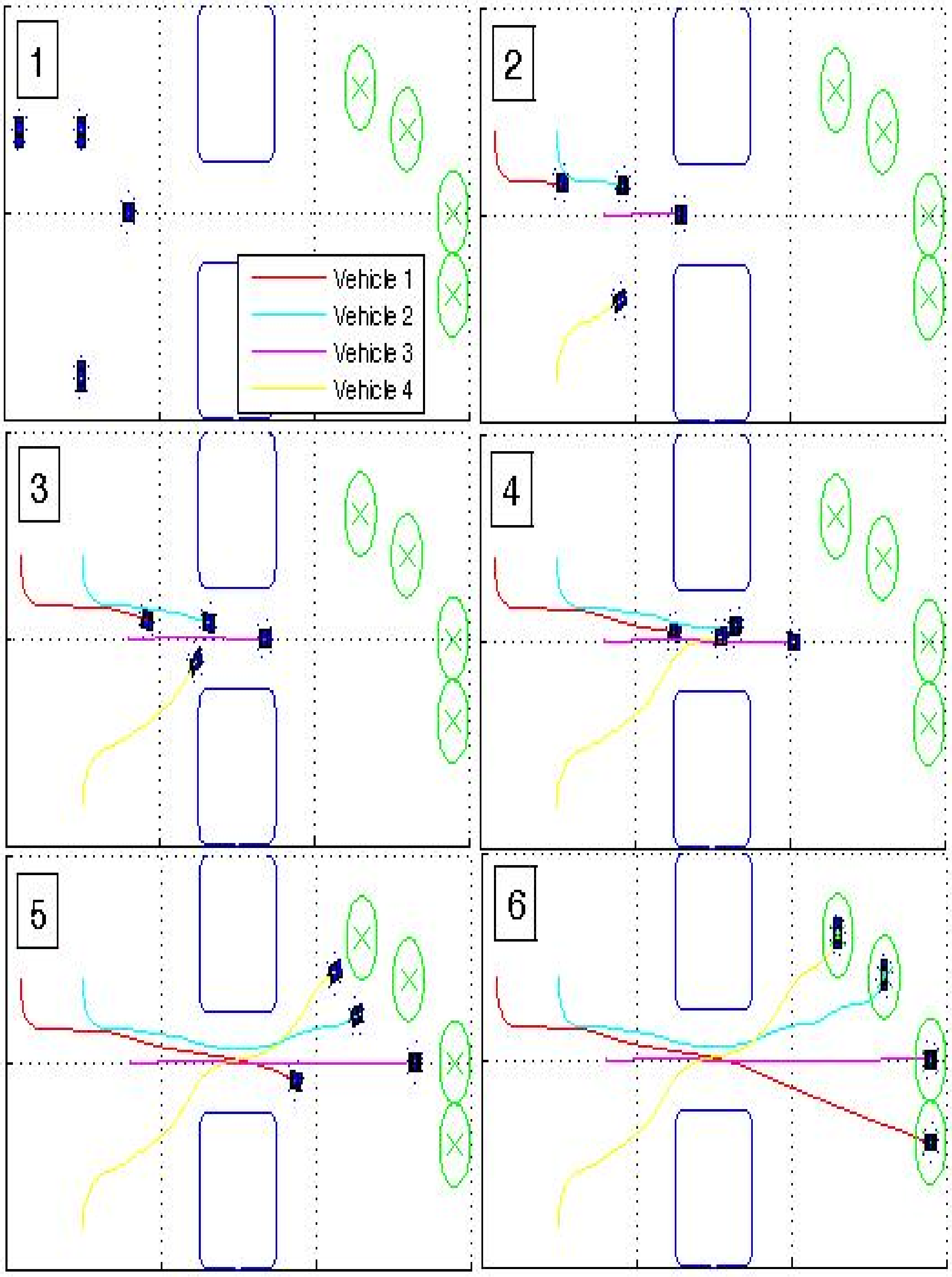


Figure 140. Evolution of the Narrow Passage Scenario (Centralized Planning).

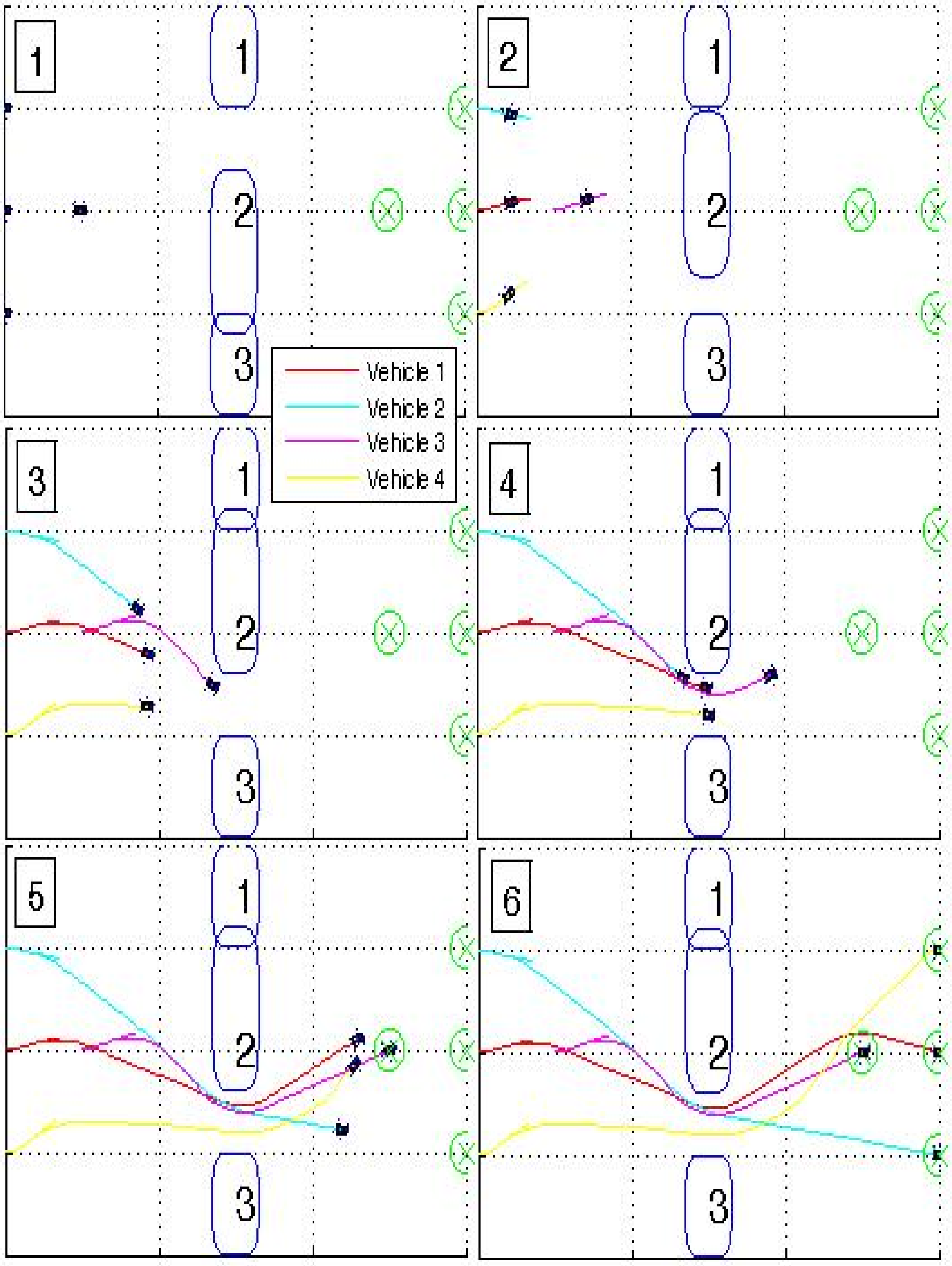


Figure 141. Evolution of the Sliding Door Scenario (Centralized Planning).

Table 19 shows a summary of the data for all three scenarios. Average maneuver times were lower than all the decoupled/non-prioritized cases, and only one prioritized case had a lower average maneuver time (sliding door // a prior // vehicle 2 prioritized). To wit, centralized planning (on average) provides the smoothest and fastest trajectories, but at the expense of high run times.

One more disadvantage to centralized planning is the fact that if one vehicle trajectory triggers a replan, then all four vehicles will replan, because they operate as one system. Although Table 19 indicates all four vehicles conducted a replan in the sliding door case (frame 2 of Figure 141), it was only the trajectory of vehicle 2 that was infeasible.

Table 19. Summary of Simulations using Centralized Planning.

Parameter		Four Corners	Narrow Pass	Sliding Door
Maneuver Time (*s*) // Corr Vehicle				
	Low	30.9 // 3	22.1 // 4	29.9 // 3
	High	31.8 // 4	30.7 // 1	42.0 // 4
	Average	31.2	24.9	38.1
Number of Re-Plans	Vehicle 1	0	0	1
	Vehicle 2	0	0	1
	Vehicle 3	0	0	1
	Vehicle 4	0	0	1
Ave Run Time (*s*) // Runs Over *50s* (*%*)		293.5 // 52.0	298.1 // 55.4	361.0 // 62.6
Min. // Max. Run Time (*s*)		5.5 // 1485.2	2.0 // 4799.3	6.8 // 1976.3

E. MULTI-VEHICLE FORMATIONS

Path planning in formation has been studied using the APF method for multiple robots [18, 19], spacecraft [20] and AUVs [21]. The use of optimal control techniques to conduct path planning for multiple vehicles in formation has not been done, to the knowledge of this author.

This section studies a line abreast formation of four vehicles. Vehicle 2 is the lead vehicle, with all other vehicles taking their position relative to vehicle 2. Vehicle 2 is also the priority vehicle, meaning that it does not include the other vehicles in its trajectory planning formulation. Vehicle 1 must position itself at a distance of *5m* from the lead vehicle at a relative bearing of 90 degrees. Relative bearings are taken with respect to the lead vehicle's heading. Vehicle 3 must position itself at a distance of *5m* from the lead vehicle at a relative bearing of -90 degrees. Vehicle 4 must position itself at a distance of *10m* from the lead vehicle at a relative bearing of -90 degrees. Vehicle 2 will have a more limited speed range ($-0.5 \leq v \leq 0.5$) than the other three vehicles ($-1 \leq v \leq 1$) so as to give the other vehicles the ability to maneuver as necessary to maintain formation. A formation vehicle must be able to deviate from its path for obstacle avoidance and still catch up to reacquire the formation.

Two information/coordination levels will be demonstrated (prediction and *a priori*). Snapshots cannot be used to generate trajectories in this case, because it assumes that only the instantaneous positions of other vehicles are known; and in order to maintain a relative bearing on another vehicle, more than just the position is necessary. Snapshots is too low an information level to maintain a line abreast formation. If all that was required was to maintain a line formation (not a line abreast), then the vehicles would be using true bearing from the lead vehicle (vice relative bearing), and knowing only the lead vehicle's position would suffice.

As in many other scenarios demonstrated thus far, the formulation of this problem requires only to adjust the cost function for each vehicle. Nothing new is required for the lead vehicle, which is simply conducting decoupled planning as the priority vehicle. For the vehicles required to maintain their positions on the lead vehicle (thus, maintaining the formation), the formulation is quite simple. If prediction is the level of cooperation being used, then the lead vehicle's current position (x_0, y_0), course (θ_0) and speed (v_0) are used to predict its trajectory at each time instant. Since its course can change rapidly from one time instant to another, only the first *8 sec* of the trajectory is used (corresponding again to the vehicle's caution zone). Given an *8 sec* time vector ($\underline{t}$) and start time (t_0), the lead vehicle's trajectory ($\underline{x_2}, \underline{y_2}$) can be calculated using Equation set (69):

$$\begin{aligned} \underline{x_2} &= \underline{x_0} + v_0 \cos(\theta_0)(\underline{t} - \underline{t_0}) \\ \underline{y_2} &= \underline{y_0} + v_0 \sin(\theta_0)(\underline{t} - \underline{t_0}) \end{aligned} \tag{69}$$

This *8 sec* trajectory is shifted to provide each of the other vehicles their paths to follow. The paths that vehicles 1, 3 and 4 must follow to maintain formation are given by Equation set (70):

$$\begin{aligned} \underline{x_1} &= \underline{x_2} - 5\sin(\underline{\theta_0}) \\ \underline{y_1} &= \underline{y_2} + 5\cos(\underline{\theta_0}) \\ \underline{x_3} &= \underline{x_2} + 5\sin(\underline{\theta_0}) \\ \underline{y_3} &= \underline{y_2} - 5\cos(\underline{\theta_0}) \\ \underline{x_4} &= \underline{x_2} + 10\sin(\underline{\theta_0}) \\ \underline{y_4} &= \underline{y_2} - 10\cos(\underline{\theta_0}) \end{aligned} \tag{70}$$

Each vehicle's initial conditions are its current position, course and speed. All end point conditions are left open, except time, which is set at the current time (t_0) plus *8 sec* ($t_f = t_0 + 8$). With time fixed, the vehicles that are maintaining formation on the lead vehicle are no longer conducting a minimum time problem. The cost function includes only the running cost (which includes the robustness function, r(t), and the path cost, p(t)) and no endpoint cost:

$$J[\underline{x}(\cdot), \underline{u}(\cdot)] = \int_{t_0}^{t_f} \big(r(t) + p(t)\big)\, dt \tag{71}$$

So, what is it that controls the speed of these vehicles? The lead vehicle's speed is determined by the minimum time formulation. The other vehicles' speeds are determined by the fact that the path they must each follow is already parameterized by time. The time parameterization is inserted by Equation (69).

Using Equation (71) and an open end point condition, a non-priority vehicle will follow its path (thus maintaining formation) and deviate from the path as necessary for obstacle avoidance. The problem here is in the shortness of the path (time horizon) versus the deviation necessary to avoid an obstacle. If the obstacle is small or the deviation from the path is small, the planner can solve for a valid trajectory. But when the path deviation or obstacle size become large, the vehicle can get stuck, because the end point is open and the planner determines the vehicle's lowest cost is to stay where it is. This is similar to the APF method getting stuck in a local minimum, which is possible because the trajectory being calculated is only *8 sec* long and does not extend out far enough to allow the vehicle to break free of that local minimum. Recall from [62] that as the horizon gets larger the problem approaches that of an optimal control problem, but as the horizon gets smaller (as occurs here when only planning out *8 sec*) it approaches that of an APF problem. The way to break free from the stuck position is to change the path being followed. Instead of following the shifted path of Equation (70), the vehicle will begin following the lead vehicle trajectory (un-shifted path) of Equation (69). This switch of paths is simply determined by the size of the path cost. When the path cost exceeds a threshold (i.e., the vehicle is too far off its path), it alters its formulation by switching to the un-shifted path, because that trajectory is the one that is known to be obstacle free. This is shown for vehicle 1 as Equation set (72):

$$\begin{aligned} &\underline{x_1} = \underline{x_2} - 5\sin(\underline{\theta_0}), \underline{y_1} = \underline{y_2} + 5\cos(\underline{\theta_0}) ... p(t) \le threshold \\ &\left(\underline{x_1}, \underline{y_1}\right) = \left(\underline{x_2}, \underline{y_2}\right) .... p(t) > threshold \end{aligned} \tag{72}$$

The vehicle will then follow the un-shifted path of the lead vehicle until a straight line path exists (with no obstacle collision) from the current vehicle location to the shifted path. When that condition occurs, the vehicle switches back to the shifted path.

If *a priori* is the level of cooperation being used, then the lead vehicle's entire trajectory is known (no prediction required). If the *8 sec* horizon is used, then the first *8* *sec* of the lead vehicle's trajectory becomes the un-shifted trajectory ($\underline{x_2}, \underline{y_2}$). That trajectory is then shifted using Equation set (73) to provide the paths that vehicles 1, 3 and 4 must follow to maintain formation.

$$\begin{aligned}
\underline{x_1} &= \underline{x_2} - 5\sin(\underline{\theta_2}) \\
\underline{y_1} &= \underline{y_2} + 5\cos(\underline{\theta_2}) \\
\underline{x_3} &= \underline{x_2} + 5\sin(\underline{\theta_2}) \\
\underline{y_3} &= \underline{y_2} - 5\cos(\underline{\theta_2}) \\
\underline{x_4} &= \underline{x_2} + 10\sin(\underline{\theta_2}) \\
\underline{y_4} &= \underline{y_2} - 10\cos(\underline{\theta_2})
\end{aligned} \tag{73}$$

There are no other differences between *a priori* and prediction. However, using *a priori* eliminates the need for an *8 sec* horizon. If no *8 sec* horizon is used (planning start-to-finish), the formulation looks more like an optimal control problem; therefore, the problem of getting stuck does not exist. In that case the path cost does not have to change between shifted and un-shifted paths. The shifted paths of Equation (73) can be used throughout the maneuver.

1. Lead Vehicle Following a Specific Path

In this scenario, the lead vehicle (vehicle 2) is given the same path following scenario as Figure 124. This time, however, vehicles 1, 3 and 4 are added to produce a line abreast formation. The evolution of the scenario using *a priori* is shown in Figures 142a (first leg) and 142b (second leg). Figure 143 provides the final trajectories of the vehicles if using prediction. Table 20 gives a summary of the pertinent data.

Frame 1 of Figure 142a shows the vehicles do not start in formation, but by frame 2 they have formed up as required. Frames 3, 4, 5 and 6 show the formation vehicles maintaining their relative positions on the lead vehicle as it deviates from its path to avoid an obstacle. Frames 1, 2 and 3 of Figure 142b show the formation vehicles maintaining their relative positions on the lead vehicle as it negotiates a 90 degree turn to the second leg of the mission. Frames 4 and 5 show vehicles 3 and 4 deviating from the formation, in order to avoid an obstacle in their path. Frame 6 shows the final vehicle positions.

Table 20 shows all maneuver times are the same. This makes sense since the vehicles are in formation and therefore finish at the same time. There are no replans, and most run times are significantly lower than all other multi-vehicle scenarios to this point. Zero replans and low run times can be attributed to the fact that a short horizon (*8 sec*) is being used to conduct the trajectory planning. Both methods (prediction and *a priori*) provide very similar results, which show them both to be effective in multi-vehicle formation scenarios.

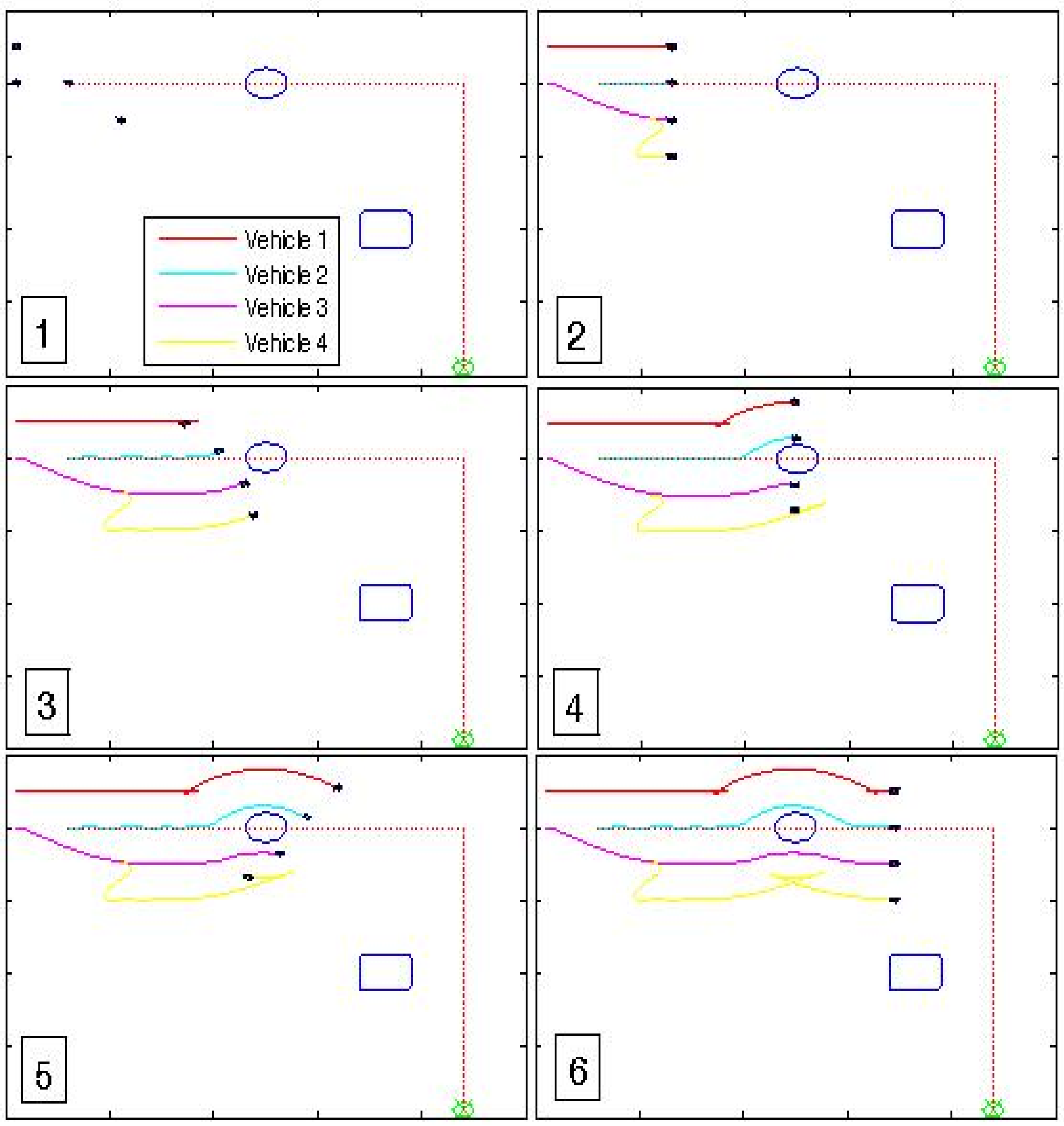


Figure 142a. 1st Leg of Formation Scenario, Lead Vehicle Path Following (*a priori*).

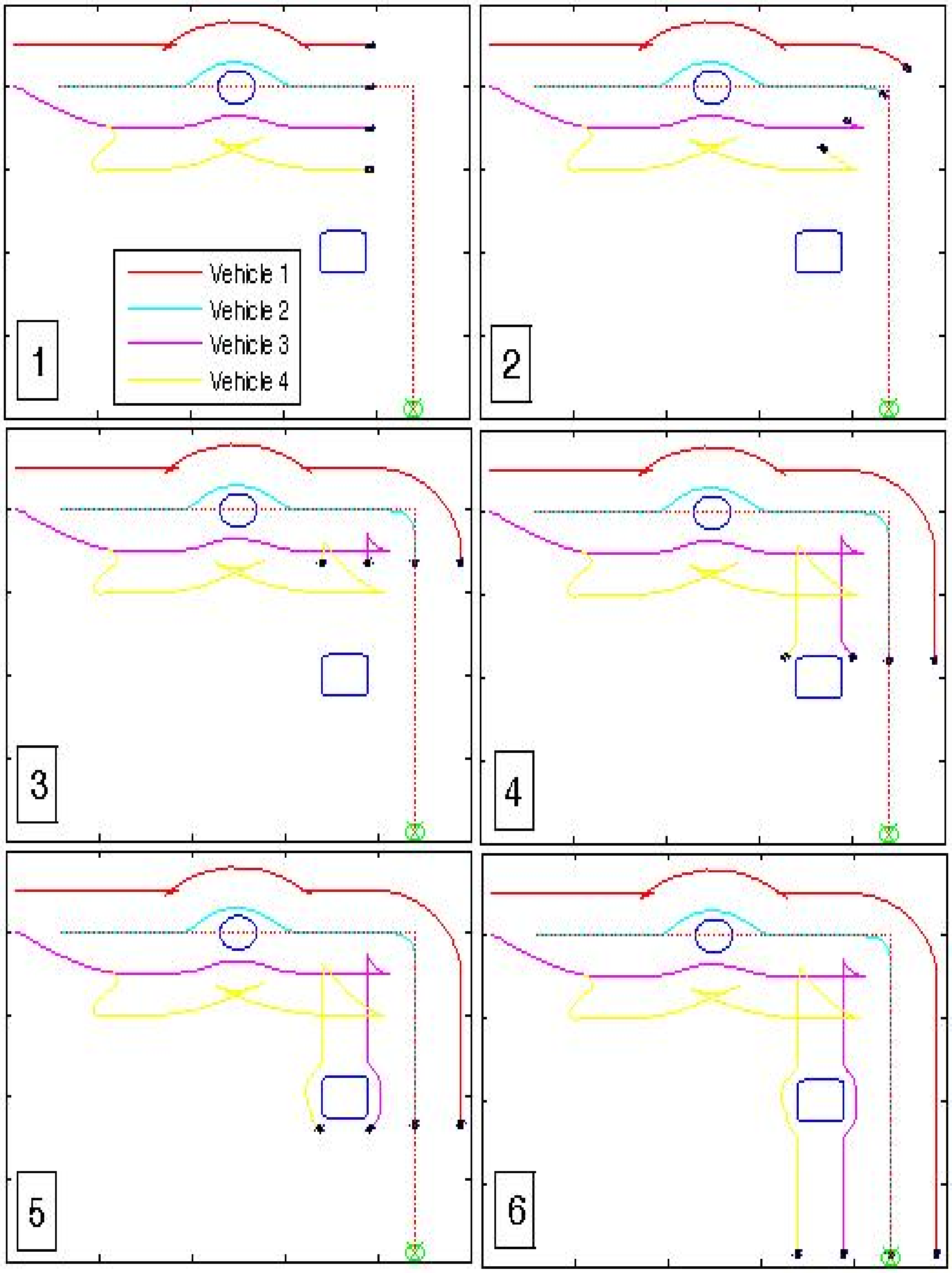


Figure 142b. 2nd Leg of Formation Scenario, Lead Vehicle Path Following (*a priori*).

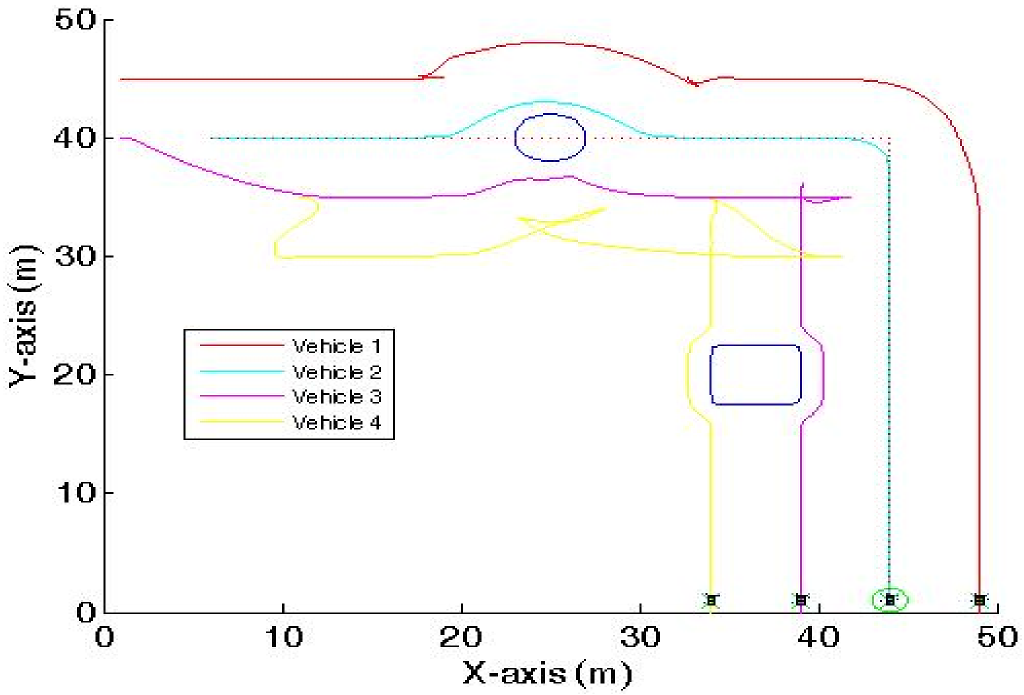


Figure 143. Formation Scenario, Lead Vehicle Path Following (Prediction).

2. Narrow Passage Scenario

In this scenario, the lead vehicle (vehicle 2) is conducting normal trajectory planning; not path following as in the previous scenario. The remaining three vehicles must form a line abreast with the lead vehicle as described at the beginning of Chapter VII.E. The evolution of the scenario using *a priori* (*8 sec*) is shown in Figure 144. Figure 145 provides the final trajectories of the vehicles if using prediction. Table 21 gives a summary of the pertinent data.

In frame 3 of Figure 144, the path cost of vehicle 1 has reached its threshold, causing the vehicle to follow the un-shifted path of the lead vehicle. By frame 4, vehicle 4 has also switched to following the lead vehicle's path. Frame 5 shows all four vehicles have negotiated the narrow passage, and vehicles 1 and 4 have re-acquired the shifted path of the lead vehicle, which causes them to speed up and maneuver to regain formation (frame 6). Frame 5 also shows the point at which vehicle 1 is performing its replan, which occurs due to a collision detection of its trajectory with the lead vehicle.

Table 20. Summary of Formation Scenario, Lead Vehicle Path Following.

Parameter	Prediction	*a priori*
Maneuver Time (*s*) // Corresponding Vehicle		
Low	155.6 // all	155.6 // all
High	155.6 // all	155.6 // all
Average	155.6	155.6
Number of Re-Plans		
Vehicle 1	0	0
Vehicle 2	0	0
Vehicle 3	0	0
Vehicle 4	0	0
Average Run Time (*s*) // Runs Over *50s* (*%*)		
Vehicle 1	1.5 // 0	1.5 // 0
Vehicle 2	1.3 // 0	1.5 // 0
Vehicle 3	1.5 // 0	2.0 // 0.3
Vehicle 4	1.6 // 0	1.9 // 0
Min. // Max. Run Time (*s*)		
Vehicle 1	0.4 // 41.7	0.6 // 12.1
Vehicle 2	0.3 // 17.7	0.3 // 21.2
Vehicle 3	0.6 // 9.3	0.6 // 174.3
Vehicle 4	0.6 // 11.1	0.7 // 18.1

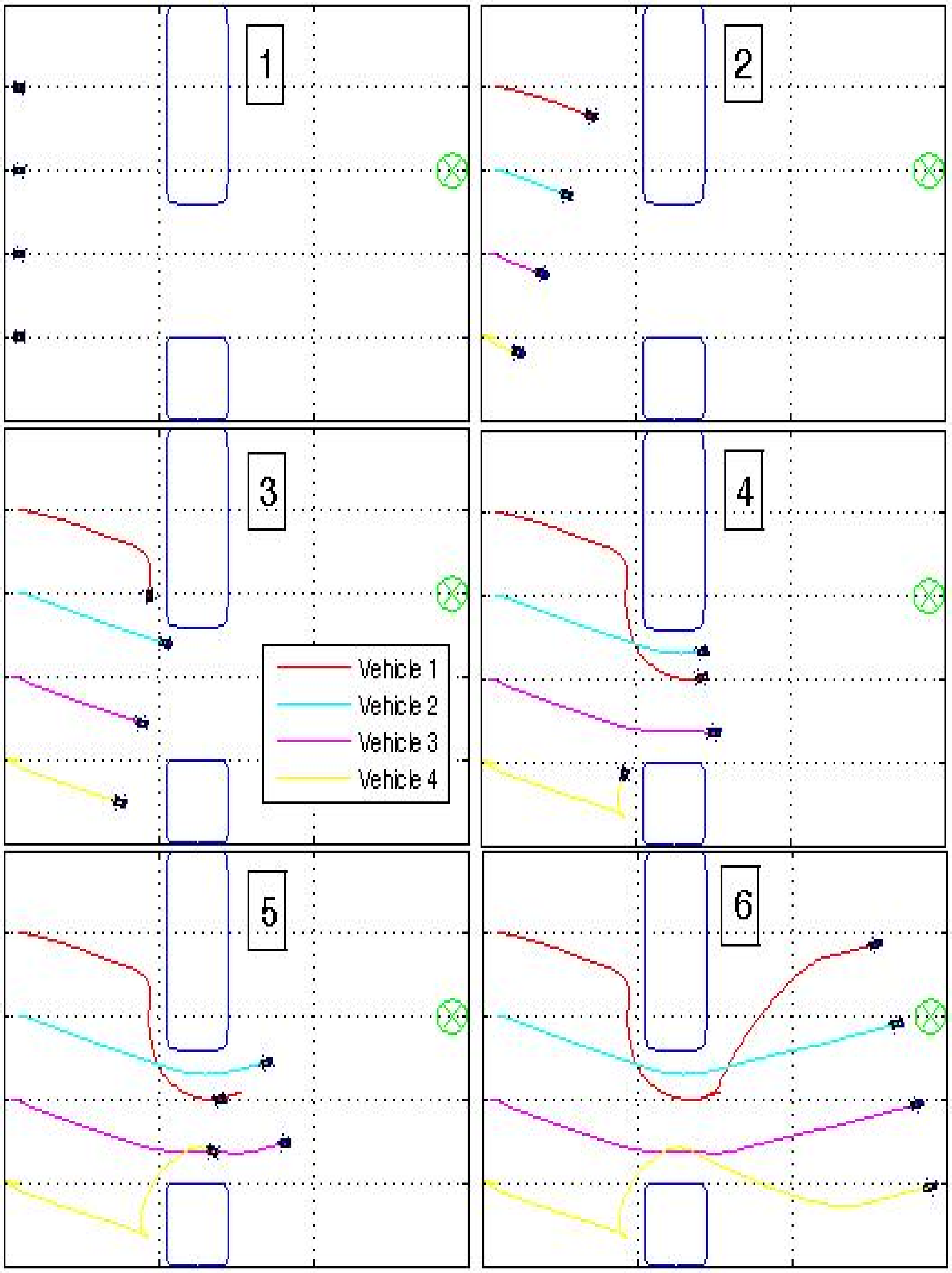


Figure 144. Evolution of the Narrow Passage Scenario, *a priori* (8 sec).

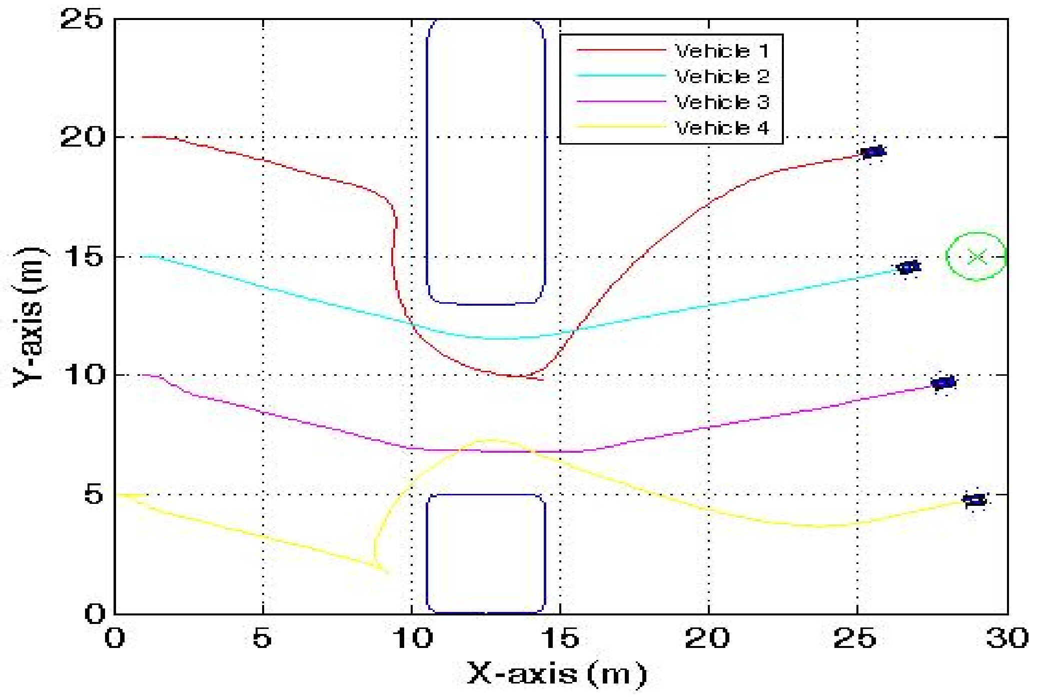


Figure 145. Narrow Passage Scenario (Prediction).

Table 21 shows all maneuver times are the same, in keeping with the fact that the vehicles are in formation. The data shows very low run times for prediction and *a priori (8 sec)*, which is again due to the short horizon over which the formation vehicles are planning. Comparing Figures 144 and 145, and observing the data in Table 21, shows that these two methods provide very similar results.

It was stated earlier that when using *a priori*, the short horizon is not needed, allowing start-to-finish planning. Figure 146 shows the evolution of the narrow passage scenario when using *a prior* (start-to-finish). Frame 3 shows that vehicle 1 has stopped and is replanning. The replan happens before the narrow passage due to a detected infeasibility, which then allows better timing versus the other vehicles going through the passage. The entire maneuver was conducted without shifting the paths of vehicles 1 and 4, because no local minimum problem exists when planning start-to-finish. The disadvantage is the significantly higher run times.

Table 21. Summary of Narrow Passage Scenario.

Parameter	Prediction	*a priori* (8 sec horizon)	*a priori* (start-to-finish)
Maneuver Time (*s*) // Corresponding Vehicle			
Low	60.0 // all	60.0 // all	60.0 // all
High	60.0 // all	60.0 // all	60.0 // all
Average	60.0	60.0	60.0
Number of Re-Plans			
Vehicle 1	1	1	1
Vehicle 2	0	0	0
Vehicle 3	0	0	0
Vehicle 4	0	0	0
Average Run Time (*s*) // Runs Over *50s* (*%*)			
Vehicle 1	2.2 // 0.0	2.3 // 0.0	27.3 // 23.2
Vehicle 2	3.5 // 0.0	3.4 // 0.0	2.5 // 0.0
Vehicle 3	1.9 // 0.0	1.8 // 0.0	22.4 // 19.6
Vehicle 4	2.1 // 0.0	2.4 // 0.0	24.1 // 13.6
Min. // Max. Run Time (*s*)			
Vehicle 1	0.9 // 9.6	0.9 // 8.5	1.7 // 146.6
Vehicle 2	0.5 // 8.8	0.6 // 8.8	0.5 // 8.9
Vehicle 3	0.7 // 21.5	0.7 // 13.6	3.4 // 106.4
Vehicle 4	0.8 // 8.7	1.0 // 8.6	2.8 // 84.9

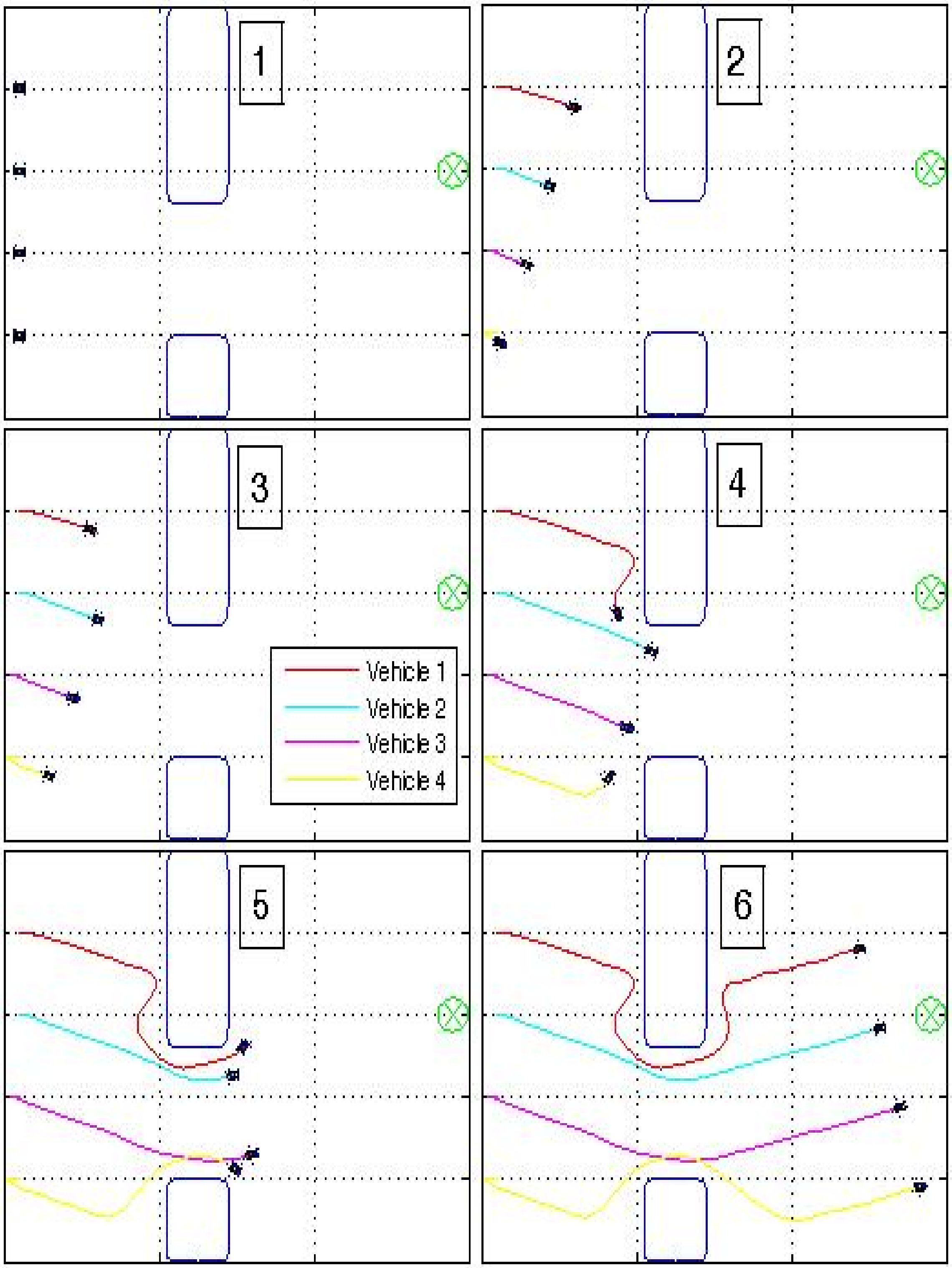


Figure 146. Evolution of the Narrow Passage Scenario, *a priori* (start-to-finish).

F. SECTOR KEEPING // PURSUIT

Maintaining a ship in its sector is very common in fleet operations in the Navy. The idea is to keep your ship in its designated sector relative to a reference point, which is usually a high value unit, such as an aircraft carrier or large deck amphibious ship. This sector keeping concept can be applied to vehicles as well. A pursuit is a special kind of sector keeping. Pursuit involves keeping another vehicle at a certain distance from the lead vehicle, but not getting too close. Therefore, a pursuit is simply a 360 degree sector. The sector keeping scenario in this section requires vehicles 1, 3 and 4 to maintain a 360 degree sector at a range of three to five meters from the lead vehicle (vehicle 2).

The formulation of this problem requires only to adjust the cost function for each pursuing vehicle. As in the formation problems, time is fixed to an *8 sec* horizon, resulting in the same cost equation as Equation (71). The difference between formation keeping and sector keeping is in the formulation of the path cost, *p(t)*. Using (x_{sk}, y_{sk}) as the sector keeping vehicle's position, (x_l, y_l) as the lead vehicle's position, and *d* as the distance between the two, the path cost becomes:

$$\begin{aligned} p &= -\ln\left[\left(\frac{x_{sk}-x_l}{3}\right)^2+\left(\frac{y_{sk}-y_l}{3}\right)^2\right]..........d \le 3 \\ p &= 0...3 < d < 5 \\ p &= \ln\left[\left(\frac{x_{sk}-x_l}{5}\right)^2+\left(\frac{y_{sk}-y_l}{5}\right)^2\right]...........d \ge 5 \end{aligned} \tag{74}$$

The path cost can be seen graphically in Figure 147. The natural log provides the scaling needed to balance the path cost with the robustness cost. In establishing the minimum cost, each pursuing vehicle will seek a position off the lead vehicle that is three to five meters away from it.

Figure 148 shows the path of the lead vehicle and the environment for the sector keeping scenario. There are eight randomly placed obstacles. The lead vehicle travels along the indicated path by breaking up its trajectory into segments, as described in Chapter VI.D. Essentially, this breaks up its maneuver from start to finish, using the five waypoints and the finish point, into six separate trajectory planning problems. As the lead vehicle nears the goal (waypoint) for a given segment, the goal is automatically re-assigned to the next waypoint.

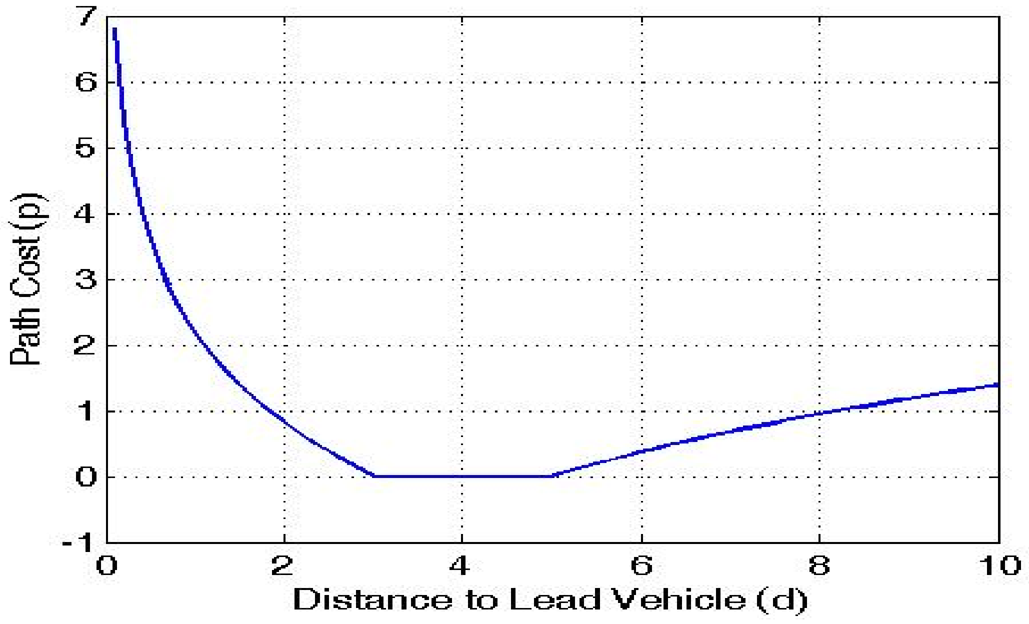


Figure 147. Path Cost vs. Distance to Lead Vehicle.

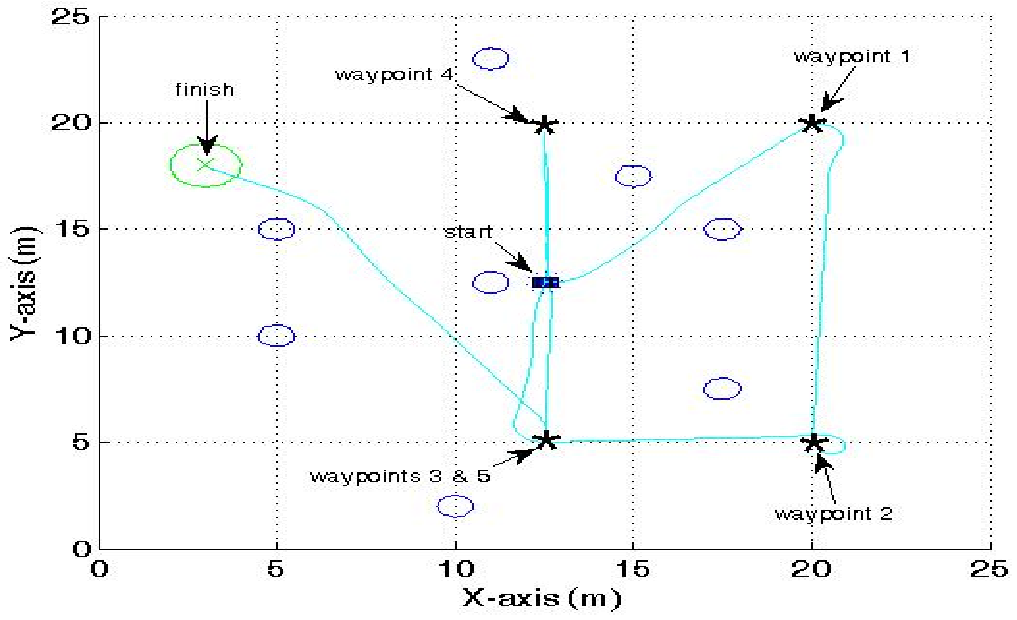


Figure 148. Lead Vehicle Trajectory for Sector Keeping Scenario.

Table 22 presents a summary of the run time data for all cooperation levels in the sector keeping scenario. The evolution of the scenario using *a priori* is shown in Figure 149. Frames 1 and 8 show the start and finish positions of all the vehicles, respectively. The dotted rings around the lead vehicle show the desired sector. Frames 2 through 7 show the positions of the vehicles at selected times on each segment of the maneuver. It can be seen that the pursuit vehicles are staying in the desired sector as the lead vehicle travels along its trajectory. Using prediction also resulted in the pursuit vehicles staying within the desired sector. However, when using snapshots, the pursuit vehicles often wandered out of the desired sector, as can be seen in Figure 150.

Table 22. Summary of Sector Keeping Scenario.

Parameter	Snapshots	Prediction	*a priori*
Average Run Time (*s*) // Runs Over *50s* (*%*)			
Vehicle 1	6.5 // 0.2	3.0 // 0.0	4.3 // 0.0
Vehicle 2	1.5 // 0.0	1.7 // 0.0	1.8 // 0.0
Vehicle 3	6.2 // 0.0	3.0 // 0.0	3.8 // 0.5
Vehicle 4	5.9 // 0.0	4.2 // 0.0	3.3 // 0.0
Min. // Max. Run Time (*s*)			
Vehicle 1	0.7 // 50.6	0.3 // 32.5	0.4 // 41.8
Vehicle 2	0.5 // 4.9	0.3 // 4.9	0.3 // 7.0
Vehicle 3	0.6 // 40.1	0.3 // 31.9	0.3 // 94.5
Vehicle 4	0.3 // 42.4	0.5 // 35.3	0.5 // 31.8

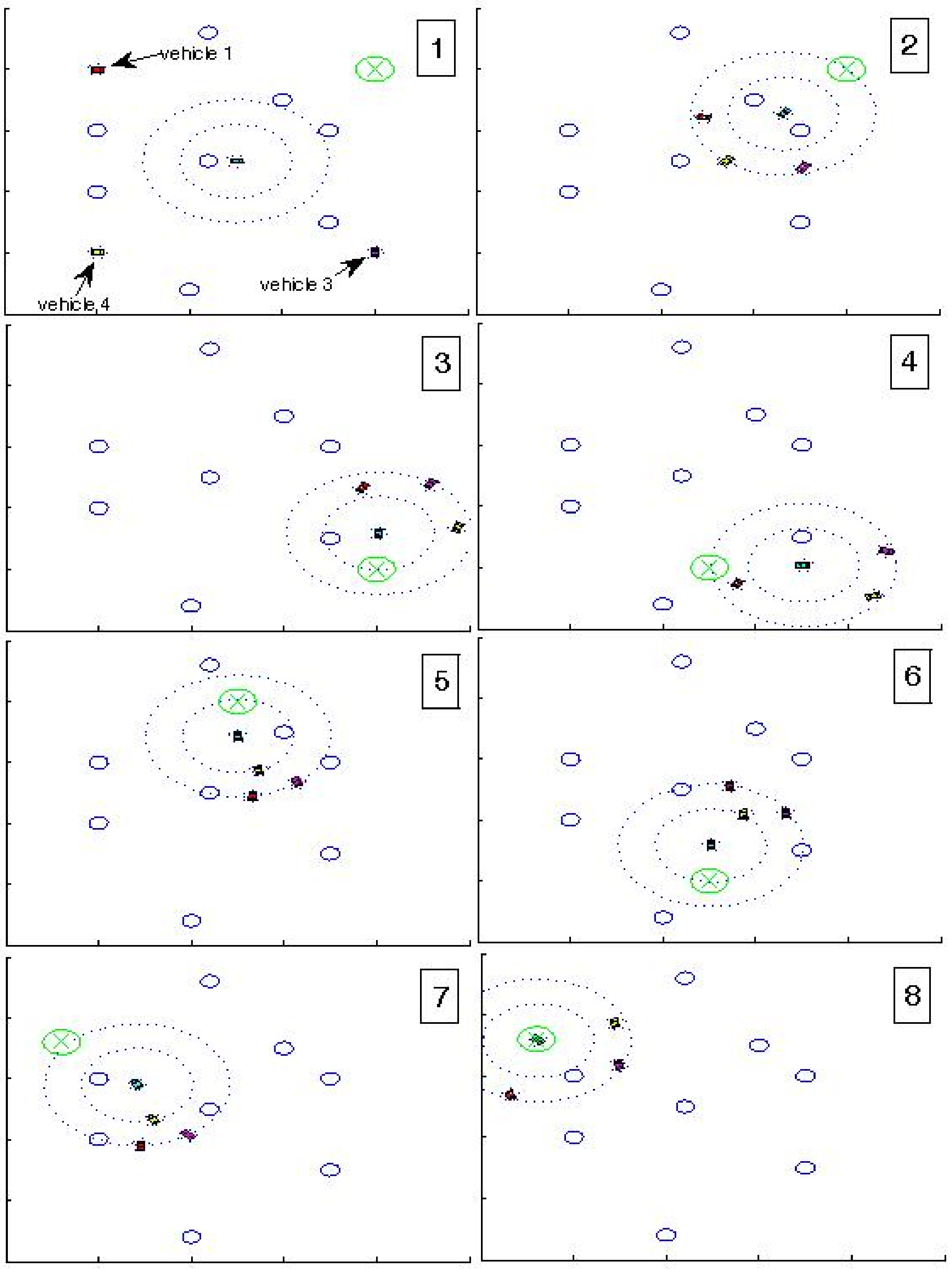


Figure 149. Evolution of the Sector Keeping Scenario (*a priori*).

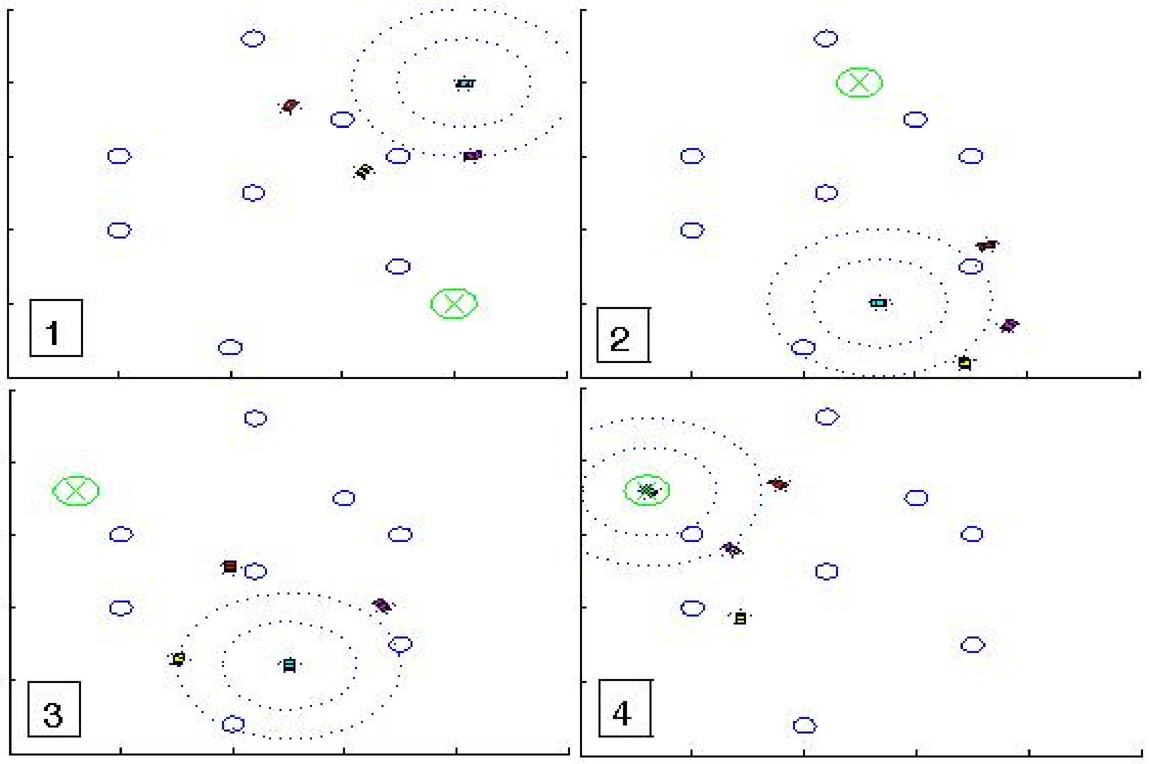


Figure 150. Selected Frames of the Sector Keeping Scenario (Snapshots).

G. CONCLUSIONS

This chapter covered numerous scenarios in an effort to determine the effect of the various information levels on multi-vehicle trajectory planning. The overall consensus is the same as that obtained in Chapter V, which is that more information is better. The more information that can be incorporated in the problem formulation; the faster the maneuver times. However, more information also results in higher run times. It has been demonstrated that the more complex the scenario (or the more information used), the more difficult it becomes to get run times under *50 sec*. Run times will always require improvement as complexity and/or information level increases.

Prioritizing a vehicle will improve its performance greatly, but there is no correlation as to the effects it will have on the other vehicles. In some cases, the performance of the other vehicles improved, but in other cases they performed worse, or their performance did not change at all.

Both prediction and *a priori* are effective cooperation levels when conducting multi-vehicle formation trajectory planning or sector keeping. Despite *a priori* having some larger run times, it can be extended beyond the *8 sec* horizon to cover the complete start-to-finish trajectory. Snapshots cannot be used for formations requiring relative bearings, and vehicles cannot be kept in their sectors at all times.

With regards to the performance criteria of Chapter II, no degradation was observed except in computational complexity, which was due to the ever increasing complexity of the scenarios, resulting in higher than desired computation times.

THIS PAGE INTENTIONALLY LEFT BLANK

VIII. CONCLUSIONS AND FUTURE WORK

A. CONCLUSIONS

The many scenarios and missions simulated in this work to study trajectory planning utilize the standard optimal control problem formulation, details of which are given in Chapter III. The contribution here is in the fact that all the changes to the problem formulation needed to produce the myriad scenarios of this work are achieved by changing the elements that make up the cost function. Changes to these elements will also cause changes to the constraints and end point conditions as well. The universal cost function for the open loop optimal control problem developed in this work is given as equation set (75):

$$J = w_{t_f} t_f + \int_0^{t_f} \left[r(t) + w_{p_1} p_1(t) + w_{p_2} p_2(t) + w_g g(t) \right] dt$$

$$r\left(w_{r_i}, n, m, h_i(t)\right) = \sum_{i=1}^{n+m} w_{r_i} \left(e^{e^{-h_i(x(t),y(t),x_{C_i}(t),y_{C_i}(t),a_i(t),b_i(t),p)}} - 1\right) \qquad (75)$$

$$p_1\left(x(t), y(t), x_d(t), y_d(t)\right) = \tan^{-1}\left[\left(x(t)-x_d(t)\right)^2 + \left(y(t)-y_d(t)\right)^2\right]$$

$$p_2\left(x(t), y(t), x_d(t), y_d(t), d_{\min}, d_{\max}, d\right) = \begin{cases} -\ln\left[\left(\dfrac{x(t)-x_d(t)}{d_{\min}}\right)^2 + \left(\dfrac{y(t)-y_d(t)}{d_{\min}}\right)^2\right] \ldots d \le d_{\min} \\ 0 \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots d_{\min} < d < d_{\max} \\ \ln\left[\left(\dfrac{x(t)-x_d(t)}{d_{\max}}\right)^2 + \left(\dfrac{y(t)-y_d(t)}{d_{\max}}\right)^2\right] \ldots\ldots d \ge d_{\max} \end{cases}$$

$$g\left(x(t), y(t), x_f, y_f, \varepsilon\right) = \ln\left(\left(\frac{x(t)-x_f}{\varepsilon}\right)^2 + \left(\frac{y(t)-y_f}{\varepsilon}\right)^2\right)$$

The weightings in the cost function, *J*, are treated as switches to either include a portion of the cost or exclude a portion. For example, if minimum time is desired to be included in the cost, then $w_{t_f} = 1$; if no minimum time component is necessary, then $w_{t_f} = 0$; if path planning or formation keeping is part of the vehicle's mission, then $w_{p_1} = 1$, else

$w_{p_1} = 0$; if the vehicle's mission includes sector keeping, then $w_{p_2} = 1$, else $w_{p_2} = 0$; if multi-vehicle trajectories are calculated in a centralized manner, then $w_g = 1$, else $w_g = 0$. A summary of these switching choices is given in Table 23.

Table 23. Summary of Cost Function Weighting Choices.

	w_g	w_{p_1}	w_{p_2}	w_{t_f}
Centralized (standard)	1	0	0	1
Decoupled (standard)	0	0	0	1
Path Following	0	1	0	1
Formation Keeping	0	1	0	0
Sector Keeping	0	0	1	0

The robustness cost, $r\left(w_{r_i}, n, m, h_i\right)$, depends on the equations defining obstacles/vehicles using the p-norm. These equations, h_i, depend on vehicle position, $x(t), y(t)$, obstacle/other vehicle position, $x_{c_i}(t), y_{c_i}(t)$, and the size/shape of each obstacle/vehicle, which is defined by $a_i(t), b_i(t), vs, p_i$:

$$h_i(x(t), y(t), x_{c_i}(t), y_{c_i}(t), a_i(t), b_i(t), p_i) = \left(\frac{x(t) - x_{c_i}(t)}{a_i(t)}\right)^{p_i} + \left(\frac{y(t) - y_{c_i}(t)}{b_i(t)}\right)^{p_i} - 1 \qquad (76)$$

for $i = 1..n + m$, where *n* is the number of obstacles and *m* is the number of vehicles.

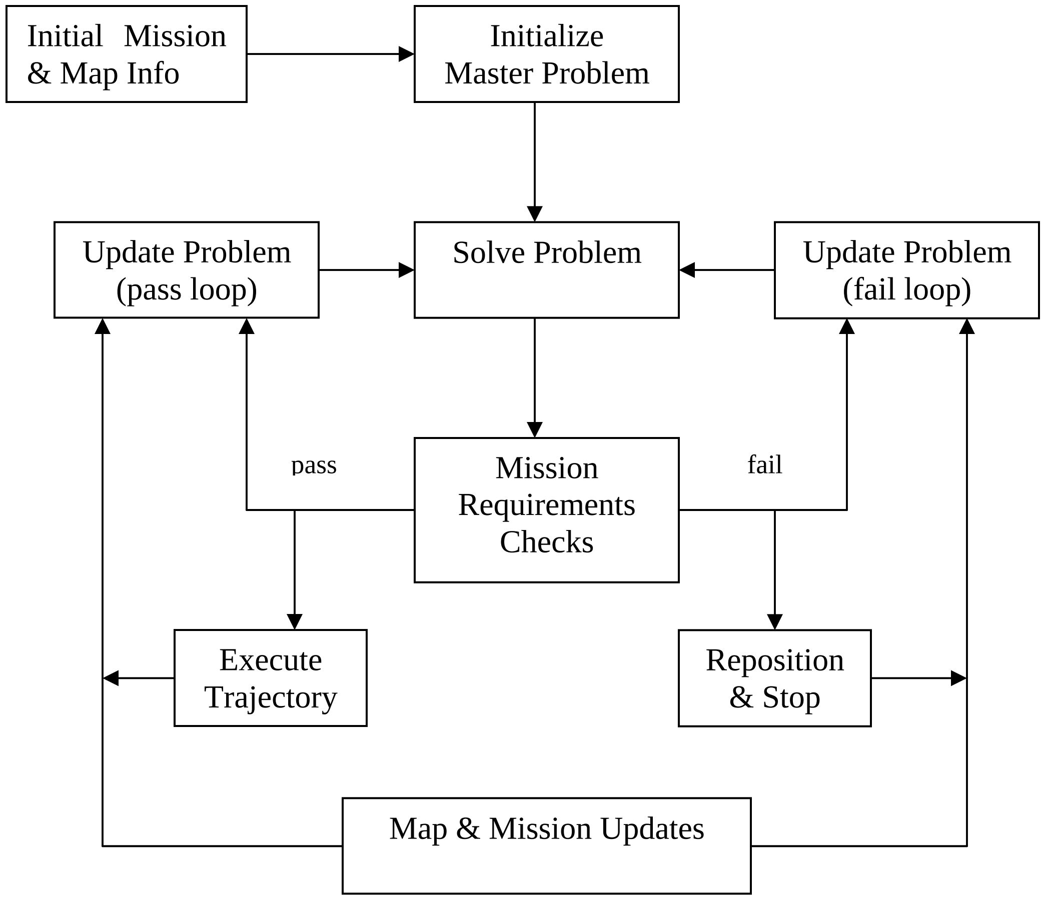
Vehicles are modeled as circular objects with $p_i = 2$ and $a_i = b_i = vs$, where *vs* is the radius of the circle. When calculating the running cost, there is a limit put on the size of p_i, which corresponds to $p_i \le 4$. This limit on p_i is explained in Chapter IV.D.2, and has the effect of accounting for the vehicle's turning dynamics while only modeling its kinematics. It should also be noted here that the same elements used to define h_i, which affects the universal cost function (*J*) through the robustness cost (*r*), are also used to define the constraint equations for the obstacles/vehicles as well. The constraint equations bear similarity to h_i, with no limit on p_i, and are shown here using *vs* as the buffer added to account for vehicle size:

$$hh_i(x(t), y(t), x_{c_i}(t), y_{c_i}(t), vs, a_i(t), b_i(t), p_i) = \ln\left[\left(\frac{x(t)-x_{c_i}(t)}{a_i(t)+vs}\right)^{p_i} + \left(\frac{y(t)-y_{c_i}(t)}{b_i(t)+vs}\right)^{p_i}\right] \quad (77)$$

for $i = 1..n+m$.

The robustness cost also depends on the number of obstacles, *n*, the number of vehicles, *m*, and the weighting, w_{r_i}, given to each obstacle/vehicle. The values of *n* and *m* used to formulate the problem may not match the actual number of obstacles/vehicles in the environment; the reason for this will become clear. Two path cost functions exist. The first path cost function, $p_1\left(x(t), y(t), x_d(t), y_d(t)\right)$, depends on the vehicle's position, $x(t), y(t)$, and the desired vehicle position, $x_d(t), y_d(t)$. The second path cost function, $p_2\left(x(t), y(t), x_d(t), y_d(t), d_{\min}, d_{\max}, d\right)$, depends on the same variables as the first path cost function, the minimum allowed distance between the vehicle and the lead vehicle, $d_{\min}$ (user specified), the maximum allowed distance between the vehicle and the lead vehicle, $d_{\max}$ (user specified), and the actual distance between the vehicle and the lead vehicle, *d*. The distance to goal function, $g\left(x(t), y(t), x_f, y_f, \varepsilon\right)$, depends on vehicle position, $x(t), y(t)$, the final endpoint condition on vehicle position, x_f, y_f, and the radius, ε, of the circle around the endpoint which the user defines as the vehicle's goal.

The basic form of the planning algorithm is presented in Figure 151. It is designed to help better understand how the basic (open loop) optimal control problem is formulated, and how changes are incorporated and fed back to the system in order to create a closed loop mechanism by which a vehicle can operate continuously in real time.



Initial Mission & Map Info
Initialize Master Problem
Update Problem (pass loop)
Solve Problem
Update Problem (fail loop)
pass
Mission Requirements Checks
fail
Execute Trajectory
Reposition & Stop
Map & Mission Updates

Figure 151. Basic Planning Algorithm.

The problem is first initialized based on the mission and known map information. This initialization sets the universal cost function of equation set (75). Built into the "Initialize Master Problem" block of Figure 151 is the decision matrix of Table 23 which determines the version of the cost function that will be utilized in the open loop problem formulation at each iteration of the algorithm. This decision matrix is shown here as Figure 152. The "Initialize w_{r_i}" block sets $w_{r_i} = w_{r_{i0}}$, where $w_{r_{i0}}$ can be chosen by the user. Keep in mind that the higher $w_{r_{i0}}$ is, the more robust the solution will be at avoiding obstacles at the expense of longer time to reach the goal. In this work, the choice was $w_{r_{i0}} = 0.2$ for circular obstacles (which includes all vehicles) and $w_{r_{i0}} = 0.15$ for all others.

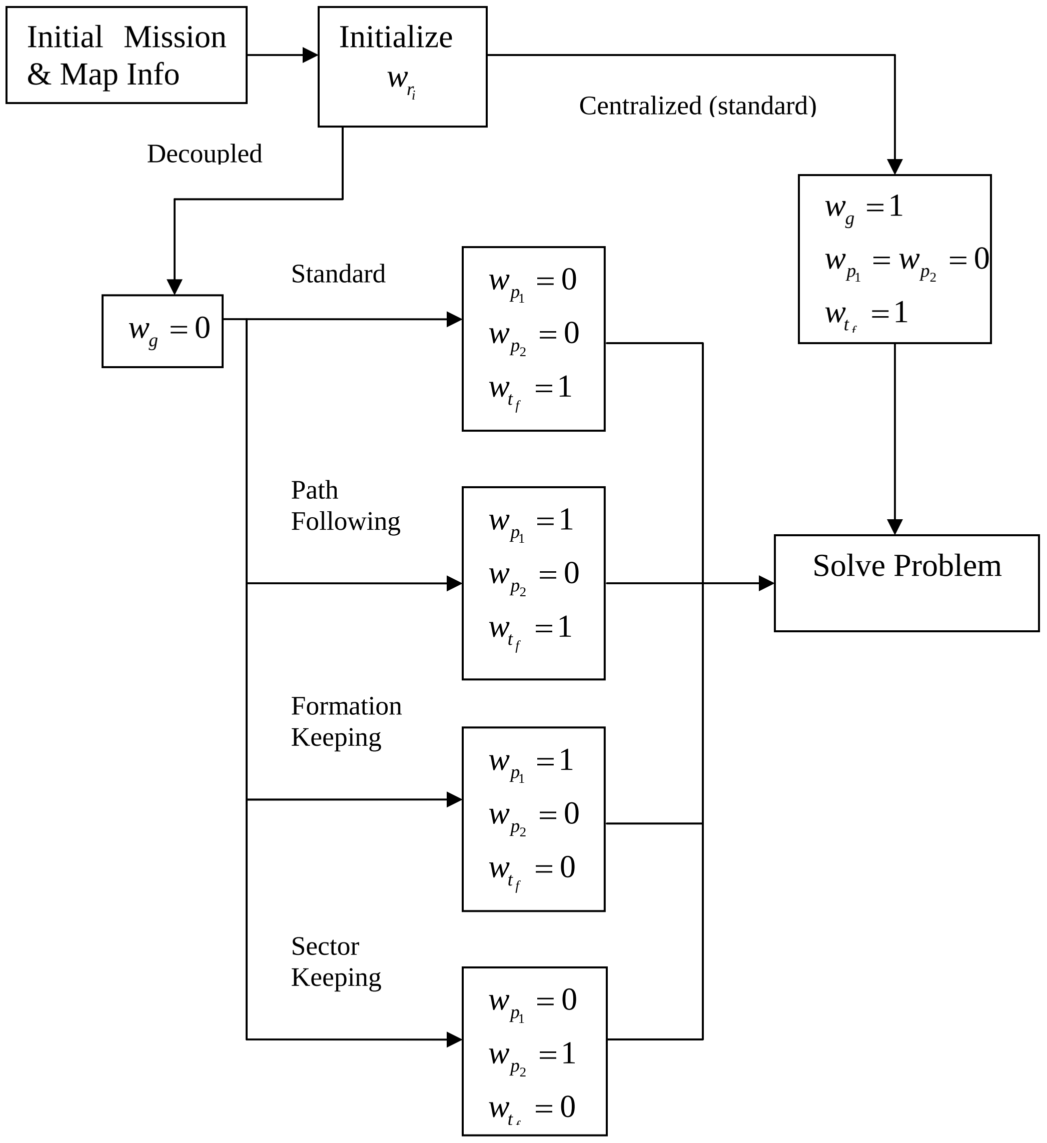


Figure 152. Initialize Master Problem Block.

After the problem is solved, various mission requirements checks are performed. Some of the checks are informative in nature, and determine the limits used to perform other checks and to update the problem once all the checks are completed. There are also checks whose outcomes are pass/fail in nature. If all the checks are passed, the "pass" loop of Figure 151 is followed. If any of the checks fail, the "fail" loop is followed. The entire algorithm can fail (i.e., get stuck in an endless fail loop) due to the map/mission update causing the solution to be physically impossible; or an algorithm failure due to bias issues. The most common task performed in the "Mission Requirements Checks" block is the conduct of collision checks. The generalized version of the constraint equation (equation (77)) is used to conduct these collision checks, but with the addition of two variables (*Z* and *sh*) that allow for the size of obstacles/vehicles to be controlled depending on the type of checks being performed. The general collision checks equation becomes:

$$col_i = \ln\left[\left(\frac{x(t)-x_{c_i}(t)}{a_i(t)+Zvs-sh_i}\right)^{p_i}+\left(\frac{y(t)-y_{c_i}(t)}{b_i(t)+Zvs-sh_i}\right)^{p_i}\right] \qquad \text{for } i=1..n+m \tag{78}$$

The mission requirements checks (from Figure 151) include:

- Feasibility checks (Chapter IV.C.4): The feasibility check is conducted at each iteration by interpolating (spline) the control solution and propagating the vehicle's path. The deviation of each propagated point from the DIDO generated solution is added up in order to establish the *norm* of the deviation. If the *norm* of the deviation exceeds a maximum value allowed by the user, *normlim*, the trajectory is not feasible, and the "fail" loop is followed (Figure 151). The equation for *norm* is:

  $$norm = \sqrt{sum\left[\left(\underline{x}(t)-\underline{xx}(t)\right)^2+\left(\underline{y}(t)-\underline{yy}(t)\right)^2\right]} \tag{79}$$

  Where $\underline{x}(t),\underline{y}(t)$ is the set of points corresponding to the vehicle's propagated trajectory at the DIDO nodes, and $\underline{xx}(t),\underline{yy}(t)$ is the DIDO generated solution. The *normlim* in this work is defined, with *N* being the number of nodes, as:

  $$norm\lim = 2N/15 \tag{80}$$

  Thus, if *norm*>*normlim*, the feasibility checks fail.

- Solid collision checks (Chapter IV.C.4): The vehicle's propagated trajectory is checked for solid collisions with any obstacle in the environment. Each point of the vehicle's trajectory is compared to each obstacle in the map using equation (78) with $Z=1$:

$$col_i = \ln\left[\left(\frac{x(t) - x_{c_i}(t)}{a_i(t) + vs - sh_i}\right)^{p_i} + \left(\frac{y(t) - y_{c_i}(t)}{b_i(t) + vs - sh_i}\right)^{p_i}\right] \quad \text{for } i = 1..n \tag{81}$$

If $col_i < 0$ for any point on the trajectory, the solid collision checks fail. sh_i is the amount each obstacle is shrunk to indicate a solid collision over a grazing collision. The determination of sh_i is as follows:

$$sh_i = G_{sh}\left(\min\left(a_i(t), b_i(t)\right) + vs\right) \tag{82}$$

The selection of $G_{sh} = 0.35$ for this work is explained in Chapter IV.D.2.

- Tight passage checks (Chapter IV.D.3): The vehicle's propagated trajectory is checked to ensure it does not pass between two obstacles that are too close together. This is done because a clearance is needed between obstacles that accounts for propagation error. This check is performed by expanding the obstacles by an additional 30% of the vehicle size (*vs*) and searching for simultaneous collisions of the vehicle's trajectory with more than one obstacle. The collision equation used to check each point of the vehicle's trajectory with all obstacles on the map is given by equation (78) with $Z = 1.3$ and $sh_i = 0$:

$$col_i = \ln\left[\left(\frac{x(t) - x_{c_i}(t)}{a_i(t) + 1.3vs}\right)^{p_i} + \left(\frac{y(t) - y_{c_i}(t)}{b_i(t) + 1.3vs}\right)^{p_i}\right] \quad \text{for } i = 1..n \tag{83}$$

  If $col_i < 0$ for any point on the trajectory simultaneously with any two or more obstacles, then the tight passage checks fail.

- Caution zone determination (Chapter IV.D.4): The caution zone is a distance chosen by the user. This distance can be defined by adjusting a gain, G_c, and is also dependant on the maximum allowed run time $(t_{\max})$, and the maximum vehicle speed ($v_{\max}$):

$$cautionzone = G_c t_{\max} v_{\max} \tag{84}$$

In this work, $G_c = 20$, $t_{\max} = 0.4$ *sec*, and $v_{\max} = 1$ *m/s* for a caution zone distance of *8m*.

- Danger zone determination (Chapter IV.D.5): The danger zone is also a distance chosen by the user. For a given $t_{\max}, v_{\max}$, this distance can be included in the algorithm by adjusting a gain, G_d:

  $$dangerzone = G_d t_{\max} v_{\max} \tag{85}$$

  In this work, $G_d = 10$ for a danger zone distance of *4m*.

- Collision checks with other vehicles (Chapter VII.B): The vehicle's propagated trajectory is checked for collision with any other vehicle in the environment within its danger zone. Each point of the vehicle's trajectory (in its danger zone) is compared to each point of the trajectories of each vehicle in the map using equation (78) with $Z = 1$ and $sh_i = 0$:

  $$col_i = \ln\left[\left(\frac{x(t) - x_{c_i}(t)}{2vs}\right)^2 + \left(\frac{y(t) - y_{c_i}(t)}{2vs}\right)^2\right] \qquad \text{for } i = 1..m \tag{86}$$

  If $col_i < 0$ for any point within the danger zone of the vehicle, then vehicle collision checks fail. If the vehicle in question is the priority vehicle, then this check ceases to be a pass/fail check, and then becomes informative in nature (Chapter VII.C). If the trajectory planning is centralized, $w_g = 1$, then vehicle collision checks are not conducted (Chapter VII.D).

- Narrow passage checks (Chapter IV.D.3): It has been determined that clearances between obstacles that are 130-200% of vehicle size require an improved resolution of the trajectory to more precisely navigate through the passage. This improved resolution is accomplished by doubling the number of nodes used to solve the problem. This check is performed within the vehicle's caution zone, only after the tight passage checks have passed, by expanding the obstacles by an additional 100% of the vehicle size (*vs*) and searching for simultaneous collisions of the vehicle's trajectory with more than one obstacle. The collision equation used to check each point of the vehicle's trajectory (in its caution zone) with all obstacles on the map is given by equation (78) with $Z = 2$ and $sh_i = 0$:

$$col_i = \ln\left[\left(\frac{x(t)-x_{c_i}(t)}{a_i(t)+2vs}\right)^{p_i} + \left(\frac{y(t)-y_{c_i}(t)}{b_i(t)+2vs}\right)^{p_i}\right] \qquad \text{for } i = 1..n \qquad (87)$$

If $col_i < 0$ for any point on the trajectory (within the caution zone) simultaneously with any two or more obstacles, then a narrow passage condition exists.

- Grazing collision checks (Chapter IV.D.2): The vehicle's propagated trajectory is checked for grazing collisions, within the vehicle caution zone, with any obstacle in the environment. This check is performed only after the solid collision checks have passed. Each point of the vehicle's trajectory (within its caution zone) is compared to each obstacle in the map using equation (78) with $Z = 1$ and $sh_i = 0$:

$$col_i = \ln\left[\left(\frac{x(t)-x_{c_i}(t)}{a_i(t)+vs}\right)^{p_i} + \left(\frac{y(t)-y_{c_i}(t)}{b_i(t)+vs}\right)^{p_i}\right] \qquad \text{for } i = 1..n \qquad (88)$$

  If $col_i < 0$ for any point on the trajectory (within the caution zone) with an obstacle, then a grazing collision condition exists with that obstacle.

- Obstacle/vehicle proximity checks (Chapter V.B.6 and VII.C): When using prediction as the information level, a moving obstacle/vehicle is only considered in the trajectory planning process if its propagated path intercepts the vehicle's caution zone (centered around its start position, (x_s, y_s), for that iteration) within the time it would take the vehicle to move to the outer edge of the caution zone at its maximum speed. For this work, the caution zone was defined as *8m*; using the vehicle's maximum speed of *1 m/s*, it would take *8 sec* for it to reach the outer edge of its caution zone. So the first *8 sec* of each obstacle/vehicle trajectory in the environment is compared to (x_s, y_s) for the vehicle whose path is being planned, using the following equation:

$$DR_i(t) = \ln\left[\left(\frac{x_s - x_{c_i}(t)}{cautionzone + vs}\right)^2 + \left(\frac{y_s - y_{c_i}(t)}{cautionzone + vs}\right)^2\right] \quad \text{for } i = 1..n+m \qquad (89)$$

If $DR_i < 0$ for any point along the path of an obstacle/vehicle, then that obstacle/vehicle is in proximity.

The "Map & Mission Updates" block (Figure 151) provides updates to the problem (cost function, constraint set, and endpoint set) regardless of whether or not the mission requirements checks pass or fail. These updates include any changes to r, p_1, p_2, and g, from equation (75), caused by changes in $n, m, a_i(t), b_i(t), x_{c_i}(t), y_{c_i}(t), x_d(t), y_d(t), d_{\min}, d_{\max}, x_f, y_f$, or ε. Also, if a vehicle is flagged as the priority vehicle, then $m = 0$ for the purpose of problem formulation for that vehicle, however, other vehicles will still be accounted for in the mission requirements checks; a vehicle will be re-added to the problem formulation if a danger zone collision is detected with that vehicle.

Built into the "Execute Trajectory" block (Figure 151) is an automatic command to stop the vehicle if a new (refreshed) trajectory is not received within the maximum allowed run time.

The "Solve Problem" block (Figure 151) uses DIDO [56] as the numerical optimization tool to rapidly generate extremals for properly formulated optimal control problems. There are some problem adjustments that occur within this block:

- When path following (Chapter VI.D): The problem of having to tune multiple running cost parameters at once is eliminated by performing a switching action on the running cost. This is achieved by using the path following portion of the running cost only (i.e., $w_{r_i} = 0$), until the vehicle feels the effects of any nearby obstacle. The influence of nearby obstacles is measured using the robustness function. When the robustness function exceeds a threshold (i.e., $r(t) \geq threshold$), the algorithm switches from path following ($w_{r_i} = 0$) to obstacle avoidance ($w_{p_1} = 0$).

- Action for formation vehicles when planning with a short time horizon (Chapter VII.E): Normally, a formation vehicle follows the shifted path of the lead vehicle. However, sometimes it is necessary to follow the un-shifted path of the lead vehicle, when the path cost at any point along the formation vehicle's trajectory exceeds a threshold (*p(t)>threshold*). The un-shifted path will be followed until a straight line path exists (with no obstacle collision) from the current vehicle location to the shifted path, at which point the vehicle switches back to the shifted path.

The "Update Problem (pass loop)" block (Figure 151) accounts for changes to the problem that apply if the "pass" loop is followed. These changes include the following:

- Determining the number of nodes, *N*, needed to solve the problem at each iteration (Chapter IV.D.1). This is done using the following equation:

$$N = 5ceil\left(\frac{dis}{\Omega \min size/5}\right) \tag{90}$$

  This provides the required number of nodes rounded up to the nearest multiple of five. *dis* is the distance the vehicle must travel from its current position to its goal along the current trajectory and is calculated, with N_c being the number of nodes in the current trajectory, using:

$$dis = \sum_{k=2}^{N_c} \sqrt{\left(x_k - x_{k-1}\right)^2 + \left(y_k - y_{k-1}\right)^2} \tag{91}$$

  ceil is a MATLAB command that rounds its argument up to the nearest integer. Ω is a ratio of nodal spread to obstacle size, defined by equation (54). *minsize* is defined by:

$$\min size = \min\left[\min\left(a_i(t), b_i(t)\right) \quad for \quad i = 1..n\right] + vs \tag{92}$$

- Doubling the number of nodes, *N*, if a narrow passage condition exists, or if a grazing collision is detected in the danger zone (Chapter IV.D.3 and IV.D.2):

$$N = 10 ceil\left(\frac{dis}{\Omega \min size / 5}\right) \tag{93}$$

This condition will clear, and the number of nodes will drop back down again, when the vehicle clears the narrow passage or begins to increase its distance from the obstacle that its trajectory grazed in the danger zone.

- Incrementing w_{r_i} to $w_{r_i} = w_{r_{i0}} + 1$ if a grazing collision is detected in the danger zone (Chapter IV.D.5).
- Increasing w_{r_i} based on the number of grazing collisions, σ_i, detected on each obstacle in the vehicle's caution zone (Chapter IV.D.2):

$$w_{r_i} = w_{r_{i0}} + 0.25\sigma_i\left(1 - e^{-p/8}\right) \tag{94}$$

- When using prediction: Removing any moving obstacles/vehicles, from the problem formulation, that do not come in proximity. This would result in $n = n - 1$ for each time an obstacle is removed from the formulation and $m = m - 1$ for each vehicle removed from the formulation (Chapter V.B.6).
- Action for a priority vehicle: Any other vehicle will be re-added to the problem formulation (*m=m+1*) if a danger zone collision is detected with that vehicle (Chapter VII.C).

The "Update Problem (fail loop)" block (Figure 151) accounts for changes to the problem that apply if the "fail" loop is followed. These changes include the following:

- Reinitializing w_{r_i} to $w_{r_i} = w_{r_{i0}}$ (Chapter IV.D.2).
- Determining the number of nodes, *N*, needed to generate a new solution that is expected to pass all the mission requirements checks (Chapter IV.C). The number of nodes to start at, N_0, can be determined by the user, along with the nodal increment, N_Δ. In this work, $N_0 = 15$ and $N_\Delta = 5$. The number of nodes being used is determined by the number of successive trips, α, around the "fail" loop:

$$N = N_0 + \alpha N_\Delta \tag{95}$$

- The old bias is discarded in favor of a new bias, which includes the vehicle initial conditions, its end point conditions, and any number of mid-point conditions selected by the user.

Because the vehicle has stopped during this loop, it has the potential to get stuck and never move because of endless failures. The endless failure may happen because of the inability of the mission checks to pass by an increase in the number of nodes or due to the mission and map update occurring to spoof the algorithm either by accident or by deliberation.

The developed algorithm (Figures 151, 152), along with equation sets (75) through (95), allow for the solving of a very large class of motion planning problems.

This work provides many contributions to the advancement of autonomous trajectory planning. Chapter II is a comprehensive comparison of the various trajectory planning methods, culminating in the conclusion that optimal control techniques are best suited, given the desired performance parameters. The most revealing results of this research is the fact that the grades given to the optimal control method, Table 1, can all be achieved simultaneously. No degradation of one performance parameter has to be suffered in order to improve another, with the exception of computational complexity. The calculation speeds of all scenarios in this work were studied closely, and the continued need for improvement was cited. This is reflected in the lower grade given to computational complexity in Table 1.

Chapter III provides a detailed study of the optimal control technique, including a demonstration of different ways to formulate the same problem, and vehicle and obstacle modeling techniques. Some very important results came out of this study. Specifically, how different vehicle orientations and endpoint constraints affect the outcome of the problem, and how to correctly model these orientations and constraints to produce desired results. The initial work that led to the development of Chapter III can be found in [73].

Chapter IV presents the development and implementation of a new information-centric PS optimal control-based algorithm for autonomous trajectory planning and control of UGVs with real-time information updates. The development of this algorithm allows future researchers to apply what has been learned in this work to real UGVs in actual field tests. It allows a UGV to take the initial planned trajectory as a starting point, use its sensors to continuously update its position and surrounding environment, and *react* to the changes by planning new (updated) trajectories. The innovations introduced in this chapter include the autonomous selection of the number of nodes used per iteration, the autonomous weighting of the running cost, and the caution and danger zone concepts. Chapter IV was born out of [74, 75].

Chapter V, which got its start in [76], utilizes the algorithm on a collection of motion planning scenarios with varying levels of information and studies the performance of the planner and the solution accuracies under these varying levels of information. The improved prediction method of Chapter V.B.6 is introduced here.

Chapter VI shows the portability of the algorithm by demonstrating its use as a path follower. It demonstrates the flexibility of the technique through its ability to be utilized simultaneously for path following and trajectory planning. The innovations introduced here include how the trajectory planning formulation is changed and utilized to combine path following and trajectory planning into one algorithm.

Chapter VII, much of which is included in [77], uses what was learned in Chapters V and VI to extend the use of the algorithm onto multiple vehicles at the same time. The quality of the solutions and the performance of the algorithm are again studied under varying levels of information. One of the big innovations here, and in previous chapters, is in the development of the scenarios themselves. Although some scenarios, such as the four corners scenario, have been used by others before, most of the scenarios in this work have not been done, but they provide very good analytical results. Another innovation here is the use of the path following concepts of Chapter VI to formulate the multi-vehicle formation problems of Chapter VII.

B. FUTURE WORK

Despite this work's many contributions to the field of trajectory planning, there are still many areas in need of attention or further focus. When a vehicle stops to replan a new trajectory, it is producing a new bias, because its algorithm has determined that the old bias was no longer feasible. The automation of determining this new bias has not been developed. The issue here is how to calculate a feasible trajectory that can be used as the new bias when only the start and endpoint conditions of the vehicle are known. To this end, how many nodes are required to generate this new bias? What possibly needs to be investigated is how to systematically provide multi-point guesses between the start and end points in order to generate a valid trajectory. Should one of the other planning methods be used to generate a valid trajectory, followed by using the optimal control technique to optimize it?

Using the current algorithm, when higher node solutions are required (such as for a narrow passage) the higher nodal density is applied uniformly to the entire trajectory. This can create higher run times than necessary if the narrow passage is only a small segment of a much longer trajectory. The desire would be to only use a higher nodal density in the vicinity of the narrow passage. Future research should focus on developing a trajectory planning formulation that allows for varying the nodal density such that higher densities can be applied only where they are needed.

Reducing run times will always be paramount to real time trajectory planning. Further study needs to be conducted in ways to reduce run times. For example, in this work, when using prediction, moving objects that did not come near the vehicle were ignored in the problem formulation. How could this concept of ignoring elements of the environment be further refined? How would this concept be extended to snapshot and *a prior* planning? What other criteria can be implemented in order to ignore more elements of the environment? After all, the more the amount of information can be trimmed down, the faster the run times will be.

This work assumes the size and shape of obstacles are known as soon as they are detected. What if an obstacle is detected, but its size and shape is not yet known? This will become increasingly important once conducting field tests using actual vehicles that sense obstacles as they move along. How does a new obstacle get modeled, and how is that model adjusted as the vehicle learns more about it?

A failure analysis should be conducted to determine the effectiveness of the algorithm at handling a stuck wheel or a steering problem. What if the vehicle can only turn right?

Future research needs to focus on the addition of an energy term (v^2) to the running cost. This has numerous possible effects/advantages which need investigation. Minimizing energy would have the effect of allowing the vehicle to slow down, vice always trying to move at maximum speed. In the obstacle intercept case of Chapter V.B.4, the vehicle might travel a straighter path to the goal than in Figure 110. It might prove to be another solution to the obstacle intercept problem of Figure 113, where the vehicle is essentially tricked into moving off its path. In the narrow passage multi-vehicle scenario using prediction, it might eliminate the extra maneuvering of vehicle 4 in Figure 130b. It could possibly eliminate all the replans that were necessary in order to adjust the timing of vehicles to avoid collisions. The big disadvantage of adding an energy term is the fact that it may sacrifice time in order to better minimize energy. The result will not be time optimal. Also, adding an energy term will require it to be weighted properly against the time and robustness portions of the cost function. Too many elements to balance in one equation is not desirable.

The prediction of moving obstacles and vehicles could possibly be improved using a Kalman filter. Also, the decision on whether to handle something as a moving obstacle or a vehicle has not yet been automated. How can the motion of an obstacle be evaluated such as to produce a performance measure that triggers a switch between its treatment as an obstacle or vehicle?

Path following, multi-vehicle formations, and sector keeping/pursuit scenarios were all demonstrated utilizing decoupled trajectory planning. A centralized approach to these scenarios still needs to be studied.

Finally, all of the scenarios in this work are still in the simulation stages. The next logical step is to apply what has been developed in this work to actual UGVs in a controlled laboratory environment.

LIST OF REFERENCES

[1] L.R. Lewis, *Rapid motion planning and autonomous obstacle avoidance for unmanned vehicles*, M.S. thesis, Naval Postgraduate School, Monterey, CA, December 2006.

[2] R. Siegwart and I.R. Nourbakhsh, *Introduction to Autonomous Mobile Robots*, The MIT Press, 2004.

[3] H. Choset, et al., *Principles of Robot Motion; Theory, Algorithms, and Implementation*, The MIT Press, 2005.

[4] F. Haro and M. Torres, "A comparison of path planning algorithms for omni-directional robots in dynamic environments," paper presented at the IEEE 3rd Latin American Robotics Symposium, Santiago, Chile, October 2006.

[5] I. Kaman, E. Rimon, and E. Rivlin, "Range-sensor based navigation in three dimensions," *Proceedings of the 1999 IEEE International conference on Robotics and Automation*, Detroit, Michigan, 10–15 May 1999.

[6] R.A. Langar, L.S. Coelho, and G.H. Oliveira, "K-Bug, A new Bug approach for mobile robot's path planning," paper presented at the 16th IEEE International Conference on Control Applications, Singapore, 1–3 October 2007.

[7] E. Magid and E. Rivlin, "CautiousBug: A competitive algorithm for sensory-based robot navigation," *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, September 28–October 2, 2004.

[8] P. Vadakkepat, K.C. Tan, and W. Ming-Liang, "Evolutionary artificial potential fields and their application in real time robot path planning," *Proceedings of the 2000 Congress on Evolutionary Computation*, vol. 1, pp. 256–263, 16–19 July 2000.

[9] D.H. Kim, H.W. Lee, S. Shin, and T. Suzuki, "Local path planning based on new repulsive potential functions with angle distributions," *Proceedings of the Third International Conference on Information Technology and Applications*, 4–7 July 2005.

[10] Q. Zhu, Y. Yan, and Z. Xing, "Robot path planning based on artificial potential field approach with simulated annealing," *Proceedings of the Sixth International Conference on Intelligent Systems Design and Applications*, vol. 2, October 2006.

[11] Y. Cen, L. Wang, and H. Zhang, "Real-time obstacle avoidance strategy for mobile robot based on improved coordinating potential field with genetic algorithm," 16th *IEEE International Conference on Control Applications*, Singapore, 1–3 October 2007.

[12] J.S. Zelek and M.D. Levine, "Local-global concurrent path planning and execution," *IEEE Transactions on Systems, Man, and Cybernetics,* Part A, vol. 30, No. 6, pp. 865–870, November 2000.

[13] M. Dinham and G. Fang, "Time optimal path planning for mobile robots in dynamic environments," *Proceedings of the 2007 IEEE International Conference on Mechatronics and Automation*, Harbin, China, 5–8 August 2007.

[14] T. Urakubo, K. Okuma, and Y. Tado, "Feedback control of a two wheeled mobile robot with obstacle avoidance using potential functions," *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems,* Sendai, Japan, September 28–October 2, 2004.

[15] C.W. Warren, "Multiple robot path coordination using artificial potential fields," *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, vol. 1, pp. 500–505, 13–18 May 1990.

[16] J.S. Baras, X. Tan, and P. Hovareshti, "Decentralized control of autonomous vehicles," *Proceedings of the 42nd IEEE Conference on Decision and Control,* Maui, Hawaii, 9–12 December 2003.

[17] T. Zheng and X. Zhao, "A novel approach for multiple mobile robot path planning in dynamic unknown environment," *IEEE Conference on Robotics, Automation and Mechatronics,* December 2006.

[18] T. Balch and M. Hybinette, "Social potentials for scalable multi-robot formations," *Proceedings of the 2000 IEEE International Conference on Robotics and Automation,* San Francisco, CA, 24–28 April 2000.

[19] W. Kowalczyk and K. Kozlowski, "Artificial potential based control for a large scale formation of mobile robots," *Proceedings of the 4th International Workshop on Robot Motion and Control,* pp. 285–291, 17–20 June 2004.

[20] C.M. Saaj, V. Lappas, and V. Gazi, "Spacecraft swarm navigation and control using artificial potential field and sliding mode control," *IEEE International Conference on Industrial Technology,* pp. 2646–2651, 15–17 December 2006.

[21] Q. Jia and G. Li, "Formation control and obstacle avoidance algorithm of multiple autonomous underwater vehicles (AUVs) based on potential function and behavior rules," *Proceedings of the IEEE International Conference on Automation and Logistics,* Jinan, China, 18–21 August 2007.

[22] N.M. Amato and L.K. Dale, "Probabilistic roadmap methods are embarrassingly parallel," *Proceedings of the 1999 IEEE International Conference on Robotics & Automation*, Detroit, Michigan, May 1999.

[23] R. Kindel, D. Hsu, J.C. Latombe, and S. Rock, "Kinodynamic motion planning amidst moving obstacles," *Proceedings of the 2000 IEEE International Conference on Robotics & Automation*, San Francisco, CA, April 2000.

[24] G. Sanchez and J.C. Latombe, "Using a PRM planner to compare centralized and decoupled planning for multi-robot systems," *Proceedings of the 2002 IEEE International conference on Robotics & Automation*, Washington, DC, May 2002.

[25] T.Y. Li and Y.C. Shie, "An incremental learning approach to motion planning with roadmap management," *Proceedings of the 2002 IEEE International conference on Robotics & Automation*, Washington, DC, May 2002.

[26] S.R. Martin, S.E. Wright, and J.W. Sheppard, "Offline and online evolutionary bi-directional RRT algorithms for efficient re-planning in dynamic environments," *Proceedings of the 3rd Annual IEEE Conference on Automation Science and Engineering*, Scottsdale, AZ, 22–25 September 2007.

[27] G. Song, S. Miller, and N.M. Amato, "Customizing PRM roadmaps at query time," *Proceedings of the 2001 IEEE International Conference on Robotics & Automation*, Seoul, Korea, 21–26 May 2001.

[28] M. Kazemi and M. Mehrandezh, "Robotic navigation using harmonic function-based probabilistic roadmaps," *Proceedings of the 2004 IEEE International Conference on Robotics & Automation*, New Orleans, LA, April 2004.

[29] M. Kazemi, M. Mehrandezh, and K. Gupta, "An incremental harmonic function-based probabilistic roadmap approach to robot path planning," *Proceedings of the 2005 IEEE International Conference on Robotics & Automation*, Barcelona, Spain, April 2005.

[30] Y. Yang and O. Brock, "Adapting the sampling distribution in PRM planners based on an approximated medial axis," *Proceedings of the 2004 IEEE International Conference on Robotics & Automation*, New Orleans, LA, April 2004.

[31] H. Kurniawati and D. Hsu, "Workspace importance sampling for probabilistic roadmap planning," *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, September 28–October 2, 2004.

[32] M. Saho and J.C. Latombe, "Finding narrow passages with probabilistic roadmaps: The small step retraction method," *Proceedings of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2–6 August 2005.

[33] Z. Sun, D. Hsu, T. Jiang, H. Kurniawati, and J.H. Reif, "Narrow passage sampling for probabilistic roadmap planning," *IEEE Transactions on Robotics*, vol. 21, No. 6, December 2005.

[34] M. Wzorek and P. Doherty, "Reconfigurable path planning for an autonomous unmanned aerial vehicle," paper presented at the 2006 International Conference on Hybrid Information Technology, Cheju Island, Korea, November 2006.

[35] J. Kim, R.A. Pearce, and N.M. Amato, "Extracting optimal paths from roadmaps for motion planning," *Proceedings of the 2003 IEEE International Conference on Robotics & Automation*, Taipei, Taiwan, 14–19 September 2003.

[36] E. Frazzoli, M.A. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," *Journal of Guidance, Control, and Dynamics*, vol. 25, No. 1, January-February 2002.

[37] S.M. Lavalle and S.A. Hutchinson, "Optimal motion planning for multiple robots having independent goals," *IEEE Transactions on Robotics and Automation*, vol. 14, No. 6, December 1998.

[38] R.V. Cowlagi and P. Tsiotras, "Beyond quadtrees: Cell decompositions for path planning using wavelet transforms," *Proceedings of the 46th IEEE Conference on Decision and Control*, New Orleans, LA, 12–14 December 2007.

[39] D. Zhu and J.C. Latombe, "New heurisic algorithms for efficient hierarchical path planning," *IEEE Transactions on Robotics and Automation*, vol. 7, No. 1, February 1991.

[40] G. Conte and R. Zulli, "Hierarchical path planning in a multi-robot environment with a simple navigation function," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 25, No. 4, April 1995.

[41] D.J. Huh, J.H. Park, U.Y. Huh, and H.I. Kim, "Path planning and navigation for autonomous mobile robot," *IEEE 2002 28th Annual Conference of the Industrial Electronics Society*, vol. 2, pp. 1538–1542, 5–8 November 2002.

[42] J.P. Van Den Berg and M.H. Overmars, "Using workspace information as a guide to non-uniform sampling in probabilistic roadmap planners," *Proceedings of the 2004 IEEE International Conference on Robotics & Automation*, New Orleans, LA, April 2004.

[43] L. Zhang, Y.J. Kim, and D. Manocha, "A hybrid approach for complete motion planning," *Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, San Diego, CA, October 29–November 2, 2007.

[44] J. Rosell, C. Vazquez, and A. Perez, "C-space decomposition using deterministic sampling and distances," *Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, San Diego, CA, October 29–November 2, 2007.

[45] M. Barbehenn and S. Hutchinson, "Toward an exact incremental geometric robot motion planner," *Proceedings of the 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Pittsburgh, PA, 5–9 August 1995.

[46] F. Lingelbach, "Path planning using probabilistic cell decomposition," *Proceedings of the 2004 IEEE International Conference on Robotics & Automation*, New Orleans, LA, April 2004.

[47] F. Lingelbach, "Path planning for mobile manipulation using probabilistic cell decomposition," *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, September 28–October 2, 2004.

[48] I.M. Ross, *Lecture Notes in Control and Optimization*, Naval Postgraduate School, Monterey, CA, 2007.

[49] P. Sekhavat, A. Fleming, and I.M. Ross, "Time-optimal nonlinear feedback control for the NPSAT1 spacecraft," *Proceedings of the 2005 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, Monterey, CA, 24–28 July 2005.

[50] K.P. Bollino and I.M. Ross, "Optimal nonlinear feedback guidance for reentry vehicles," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Keystone, CO, 21–24 August 2006.

[51] K.P. Bollino, L.R. Lewis, P. Sekhavat, and I.M. Ross, "Pseudospectral optimal control: A clear road for autonomous intelligent path planning," *AIAA Infotech@Aerospace 2007 Conference and Exhibit*, Rohnert Park, CA, 7–10 May 2007.

[52] L.R. Lewis and I.M. Ross, "A pseudospectral method for real-time motion planning and obstacle avoidance," *AVT-SCI Joint Symposium on Platform Innovations and System Integration for Unmanned Air, Land and Sea Vehicles*, Florence, Italy, 14–17 May 2007.

[53] K.P. Bollino and L.R. Lewis, "Optimal path planning and control of tactical unmanned aerial vehicles in urban environments," *Proceedings of the AUVSI's Unmanned Systems North America 2007 Conference*, Washington, DC, August 2007.

[54] L.R. Lewis, I.M. Ross, and Q. Gong, "Pseudospectral motion planning techniques for autonomous obstacle avoidance," *Proceedings of the 46th IEEE Conference on Decision and Control*, New Orleans, LA, 12–14 December 2007.

[55] I.M. Ross, P. Sekhavat, A. Fleming, and Q. Gong, "Optimal feedback control: Foundations, examples, and experimental results for a new approach," *Journal of Guidance, Control, and Dynamics*, vol. 31, No. 2, March–April 2008.

[56] I.M. Ross, *A Beginner's Guide to DIDO: A MATLAB Application Package for Solving Optimal Control Problems*, Elissar Technical Report TR-711, http://www.elissar.biz, 2007.

[57] I.M. Ross and F. Fahroo, "Issues in the real-time computation of optimal control," *Mathematical and Computer Modeling*, Pergamon Publication, 2005.

[58] I.M. Ross, C. D'Souza, F. Fahroo, and J.B. Ross, "A fast approach to multi-stage launch vehicle trajectory optimization," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Austin, Texas, 11–14 August 2003.

[59] AEM Design Home of FSQP, http://www.aemdesign.com (Accessed: November 2008).

[60] Stanford Business Software Inc., http://www.sbsi-sol-optimize.com (Accessed: November 2008).

[61] Numerical Algorithms Group, http://www.nag.com (Accessed: November 2008).

[62] C.G. Henshaw, "A unification of artificial potential function guidance and optimal trajectory planning," Naval Center for Space Technology, U.S. Naval Research Laboratory, Washington, DC, 10 January 2005.

[63] I.M. Ross, *Control and Optimization: An Introduction to Principles and Applications, Electronic Edition*, Naval Postgraduate School, Monterey, CA, December, 2005.

[64] Lecture Notes in Control and Information Sciences 229, *Robot Motion Planning and Control*, by J.P. Laumond, and others, Springer, 1998.

[65] J.A. Reeds and L.A. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific Journal of Mathematics*, 145(2):367–393, 1990.

[66] L.E. Dubins, "On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics*, vol. 79, pp. 497–516, 1957.

[67] W. Kang, Q. Gong, I.M. Ross, and F. Fahroo, “On the convergence of nonlinear optimal control using pseudospectral methods for feedback linearizable systems,” *International Journal of Robust and Nonlinear Control*, vol. 17, 1251-1277, online publication, 3 January 2007.

[68] Q. Gong, I.M Ross, W. Kang, and F. Fahroo, “Connections between the covector mapping theorum and convergence of pseudospectral methods for optimal control,” *Journal of Computational Optimization and Applications*, vol. 41, No. 3, pp. 307–335, online publication, 31 October 2007.

[69] I.I. Kaminer, O.A. Yakimenko, and A.M. Pascoal, “Coordinated control of multiple UAVs for time-critical applications,” *Proceedings of the 2006 IEEE Aerospace Conference*, Big Sky, Montana, 4–11 March 2006.

[70] R. Ghabcheloo, I. Kaminer, A.P. Aguiar, and A. Pascoal, “A general framework for multiple vehicle time-coordinated path following control,” *Proceedings of the 2009 American Control Conference*, St. Louis, MO, 10–12 June 2009.

[71] R.M. Murray, “Recent research in cooperative control of multi-vehicle systems,” *Journal of Dynamic Systems, Measurement, and Control*, vol. 129, No. 5, pp. 571–583, 2007.

[72] L. Cremean, W.B. Dunbar, D. van Gogh, J. Hickey, E. Klavins, J. Meltzer, and R.M. Murray, “The caltech multi-vehicle wireless testbed,” *Proceedings of the 41st IEEE conference in Decision and Control*, Las Vegas, Nevada, 10–13 December 2002.

[73] M.A. Hurni, P. Sekhavat, and I.M. Ross, “Issues on UGV optimal motion planning and obstacle avoidance,” *Proceedings of the AIAA Unmanned…Unlimited Conference and Exhibit*, Seattle, Washington, 6–9 April 2009.

[74] M.A. Hurni, P. Sekhavat, and I.M. Ross, “Autonomous trajectory planning using real-time information updates,” *Proceedings of the AIAA Guidance, Navigation and Control Conference and Exhibit*, Honolulu, Hawaii, 18–21 August 2008.

[75] M.A. Hurni, P. Sekhavat, and I.M. Ross, “Pseudospectral optimal control algorithm for real-time trajectory planning,” *Proceedings of the 19th AAS/AIAA Space Flight Mechanics Meeting*, Savannah, Georgia, 9–12 February 2009.

[76] M.A. Hurni, P. Sekhavat, and I.M. Ross, “An info-centric trajectory planner for unmanned ground vehicles,” *First International Conference on the Dynamics of Information Systems*, Gainesville, Florida, 28–30 January 2009.

[77] M.A. Hurni, P. Sekhavat, and I.M. Ross, “Autonomous multi-rover trajectory planning using optimal control techniques,” *Proceedings of the 2009 AAS/AIAA Astrodynamics Specialist Conference*, Pittsburgh, PA., 9–13 August 2009.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, VA

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, CA

3. I. Michael Ross
   Naval Postgraduate School
   Monterey, CA

4. Wei Kang
   Naval Postgraduate School
   Monterey, CA

5. Xiaoping Yun
   Naval Postgraduate School
   Monterey, CA

6. Fotis A. Papoulias
   Naval Postgraduate School
   Monterey, CA

7. Isaac I. Kaminer
   Naval Postgraduate School
   Monterey, CA

8. Guidance, Navigation, and Control Laboratory
   Naval Postgraduate School
   Monterey, CA